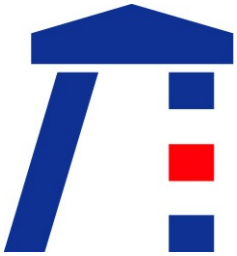


MODELING ROAD ROUGHNESS WITH CONDITIONAL RANDOM FIELDS

Alexander Lemken

Vom Fachbereich Mathematik der Technischen Universität Kaiserslautern zur Verleihung des akademischen Grades Doktor der Naturwissenschaften (Doctor rerum naturalium, Dr. rer. nat.) genehmigte Dissertation

Gutachter
Prof. Dr. Jürgen Franke (TU Kaiserslautern)
Prof. Dr. Jörn Sass (TU Kaiserslautern)

Mündliche Aussprache
26.10.2015

D 386

Acknowledgements

First and foremost I want to thank my doctoral supervisor Jürgen Franke. His support and ideas helped me shape this work. Thanks also to Jörn Sass for reviewing my thesis as well. Many thanks to Klaus Dressler and the Fraunhofer ITWM for giving me the opportunity to work on my thesis by supporting me with a Fraunhofer scholarship and a great work environment. Further I would like to thank Nikolaus Ruf, who helped me as a college at the Fraunhofer ITWM and mentored me during the work on my dissertation. In our many discussion his experience in statistics helped me to find the direction of my research.
Further I would like to thank Sebastian Seifen for being a great colleague and friend. In hours of discussion we processed many steps of my work, resulting in valuable feedback and a very good time. Also thanks to the many fellows and colleagues at the Fraunhofer ITWM. Especially the work within the Virtual Measurement Campaign team helped me to shape this work and keep it application oriented.
The biggest thanks is given to my family, who provided me with the confidence and strength to pursue this work and supported my each step of way.
Thanks.

Contents

1 Motivation

The first chapter explains the need for detailed knowledge about road quality conditions in the context of fatigue damage estimation of vehicles. We start out in section 1.1 by introducing the notion of fatigue damage, its definition and related basic concepts. In section 1.2 we discuss the concept of road roughness as a key road quality measure. In chapter 1.3 we briefly explain the Virtual Measurement Campaign and the interface to this work. Afterwards we show how a road surface can be statistically modeled. Finally in section 1.4 we make a statistical analysis of real life road quality data in order to specify a suitable class of models.

1.1 Fatigue damage of cars

The process of designing a vehicle is a highly complex as well as time and money intensive process. The customers' demands for good driving features, high comfort and high reliability have to be considered within each step of the process. The reliability of a car is highly dependent on the lack of damages. We distinguish between two kinds of damages. On the one hand we observe events which cause great damage or complete system break downs but occur very seldom. On the other hand there are small impacts that occur frequently, but do not cause a total break down. The mathematical theory used for modeling seldom occurring, but extreme events is extreme value theory. This field of is widely researched for example in [HF00]and [Sei14].
In our work we focus on the second form of damage, the fatigue damage. This term describes the forms of damage resulting from the application of a large number of small loads, which are often regarded as being periodically applied. Such loads result from the vehicle driving over uneven roads. Johannesson and Speckert [JS$^+$14] describe the influencing factors that lead to the load distribution of a vehicle in general. These factors are:

- Vehicle usage:
  - What is the driver using the vehicle for?
  - How is the vehicle used?
- Load environment:
  - What are the surrounding conditions ?
  - For example road condition, traffic, legislation, etc.
- Vehicle dynamics:
  - How is the vehicle set up?
  - What is the mechanical response to driver and environment?

A detailed introduction and overview over many techniques used to analyze and quantify fatigue damage in general and in the specific case of load analysis of vehicles can be found in [JS+14]. In our studies we focus on the analysis of the load environment. When analyzing the fatigue damage of a vehicle the key environment aspect is the interaction with the surface the vehicle moves on. In general we assume that the vehicle is operated on solid ground. The vehicle loads that we are focusing on are induced by vertical displacements of the surface. Further assuming that the car is mainly operated on a road we turn to the notion of road roughness, which is closely related to the vertical displacement.

1.2 Road roughness

Road roughness in general describes the unevenness of a road. In order to define roughness as a mathematical quantity we begin by defining road profiles.

1.2.1 Road profiles

A one-dimensional road profile is a series of elevations of a road's surface along a given path on the road. As depicted in Figure 1.1 profiles can be measured in longitudinal or lateral direction. As in many applications this work focuses on the longitudinal profiles and the roughness in longitudinal direction. This simplification is strongly motivated by the idea that the roughness' impact on the car is mainly determined by the vertical input for the car while driving in longitudinal direction. Such road profiles are not uniquely defined for a given road. Measuring a profile with different profiling techniques or at slightly different lateral positions (as depicted in Figure 1.1) will result in a different series of elevations. To compare different road profiles one can use roughness indicators, which we introduce in the next section.

It is often useful to represent a profile in the form of a spectral density (PSD), which is closely related to the Fourier spectrum of the vertical elevations. The power spectral density can be calculated using the expected value of the square of the truncated Fourier transform of the signal. This results in a function that shows the power value (squared Fourier density) in dependency of a frequency or wavelength (or wave number). A typical profile in vertical displacement notation and PSD is shown in Figures 1.2 and 1.3. Usually when looking at the PSD of a profile, we are not interested in all available frequencies but in a range of frequencies that are relevant for our cause. This can be achieved by applying frequency filters. There are many ways to either filter while measuring the profile by mechanical filter systems or as more commonly performed as a post processing with a signal-processing-filter. Typical examples are high-pass filters to capture high frequency responses that can be relevant for noise and vibration analysis or low-pass filter which deliver information for example about the hilliness of the road.

Figure 1.1: Road profiles in longitudinal and lateral direction (source [Say98])

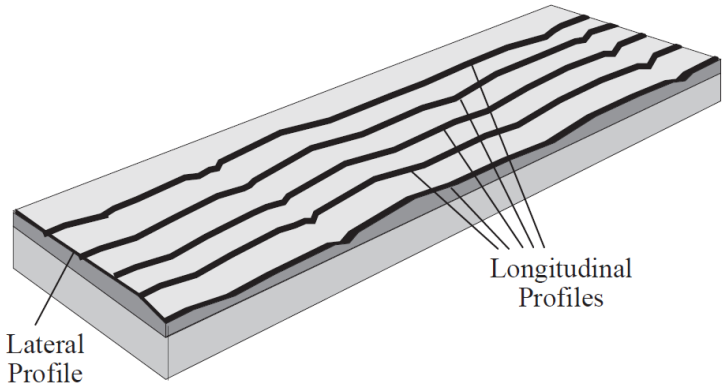


Figure 1.2: Road profile as elevations over distance (source [Say98])

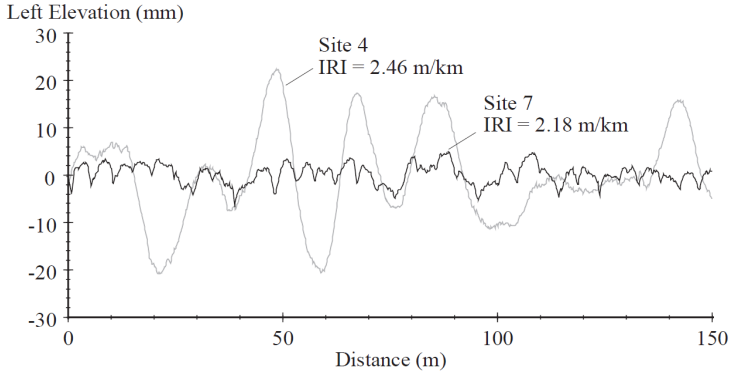
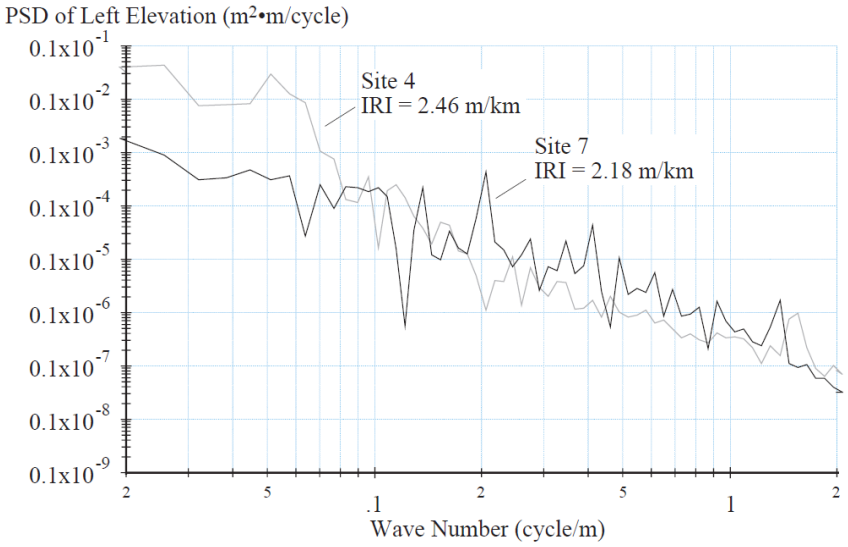


Figure 1.3: Road profile representation as PSD (source [Say98])



1.2.2 Roughness indices

The term road roughness is used for the deviation of a road profile from a perfectly planar surface. Usually only those deviations with effect for vehicles are taken into account. As mentioned above, the profile can be represented by its PSD. Reducing the roughness to the deviations that effect the vehicle or the driver is achieved by filtering the original signal, since the unfiltered spectrum can have very wide frequency support. Basing on the original definition of the road roughness there are several definitions for summary statistics on sections of a road that serve as index for the road roughness. Those indices are required to be

- Uniquely determined for a road section, if the true profile is known
- Independent of the profiling device and the specific environment conditions

According the 'The little book of profiling', [Say98] all PSD based roughness indices can be defined by the four definition steps:

1. Define how many profiles are needed for the index calculation
2. Apply a suitable filter on the PSD
3. Accumulate the profile to a single score
4. Definition of a unit and scaling of the score

There are several standardized roughness indices. We will introduce and focus on two very widespread internationally recognized indices in the following.

1.2.3 IRI

A very widespread roughness index is the International Roughness Index or IRI. It focuses on the street's effect on the driver and the vehicle and is calculated using a single one-dimensional road profile. The signal filter is defined by the principle of a multibody system called the quarter vehicle, which resembles one forth, or one corner of a vehicle. The mathematical model is a system of differential equations derived from a dynamic system of masses, springs and dampers as depicted in Figure 1.4. Using different parameters for the masses, spring and damper parameters resemble different vehicle set-ups. One particularly popular set of coefficients is known as the 'Golden Car' parameter set. This particular filter is used in the definition of the IRI. The corresponding transfer function is also depicted in figure 1.4 on the bottom left. This particular transfer functions, which defines the PSD filter, effectively discards information below wavelength of 1.2m and above 30m.

There has been an extensive study in 1982 by the world bank called 'The International Road Roughness Experiment' [Say86], which conducted a variety of experiments in order to find a well suited roughness indicator. Their experiments led to the final definition of the IRI, which was first introduced under the notion RARS80 (Reference Average Rectified Slope at 80km/h). The practical steps to acquire the IRI of a given profile are:

1. Low pass filter: A moving average filter with 250mm base length
2. Quarter car filter: The golden car filter with travel speed 80km/h
3. Average over the absolute slopes along the profile L :

$$\text{IRI} := \frac{1}{|L|} \int_L |\dot{y}(x)| dx \tag{1.1}$$

As the world bank encourages the use of IRI and describes how to retrieve the most reliable IRI values using a wide variety of profiling devices, the IRI is widely accepted by many researchers and is used in many datasets throughout the world. Very often the data is either available directly as IRI or in a closely related quantity.

Figure 1.4: The quarter vehicle model (source [Say98])

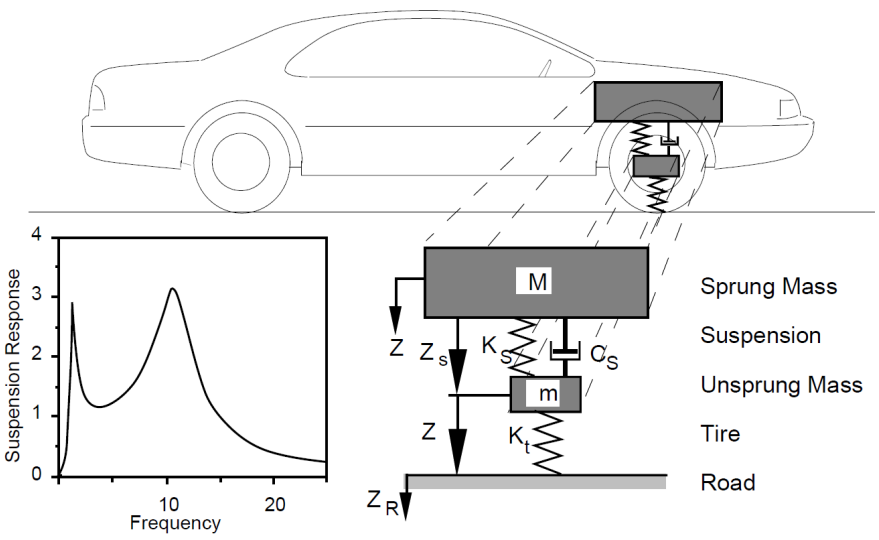
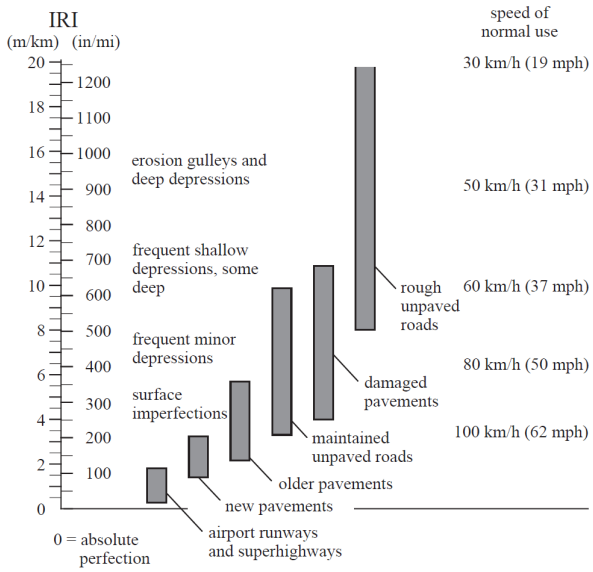


Figure 1.5: Interpretation of the IRI (source [Say98])



1.2.4 ISO specification

Another standardized roughness index is described in the ISO 8606 - 1995 ([Sta95]) definition. The idea of the index is to use the fit of a parametric function to the measured profile's PSD. This parametric function has the form

$$S_Z(\Omega) = C \cdot \left(\frac{\Omega}{\Omega_0}\right)^{-w}, \quad \text{with} \quad \Omega \in [2\pi \cdot 0.011, 2\pi \cdot 2.83] \qquad \text{and 0 otherwise} \qquad (1.2)$$

The Ω is the spatial angular frequency, which keeps the formulation independent of speed or time. Ω_0 is a reference frequency and is set to 1 rad/m by definition. The parameters are the roughness coefficient C and the waviness w. The ISO standard fixes $w = 2$ and this yields a one parameter (C) representation of an approximated PSD. This approximation is invalid for very small or large spatial angular frequencies. Therefore the analysis and the support is limited to the domain of $\Omega \in [2\pi \cdot 0.011, 2\pi \cdot 2.83]$.

The use of the ISO 8608 index is popular due to its simplicity. Furthermore it can be related to the IRI. This relation is described in detail in [JPR14]. The idea is to model a road by a Gaussian process, which exactly matches the PSD specified by a given ISO index C. This is a probabilistic approach where calculating the expected value for the IRI yields:

$$\mathbf{E}\left[\text{IRI}\right] = 2.21\sqrt{C}$$

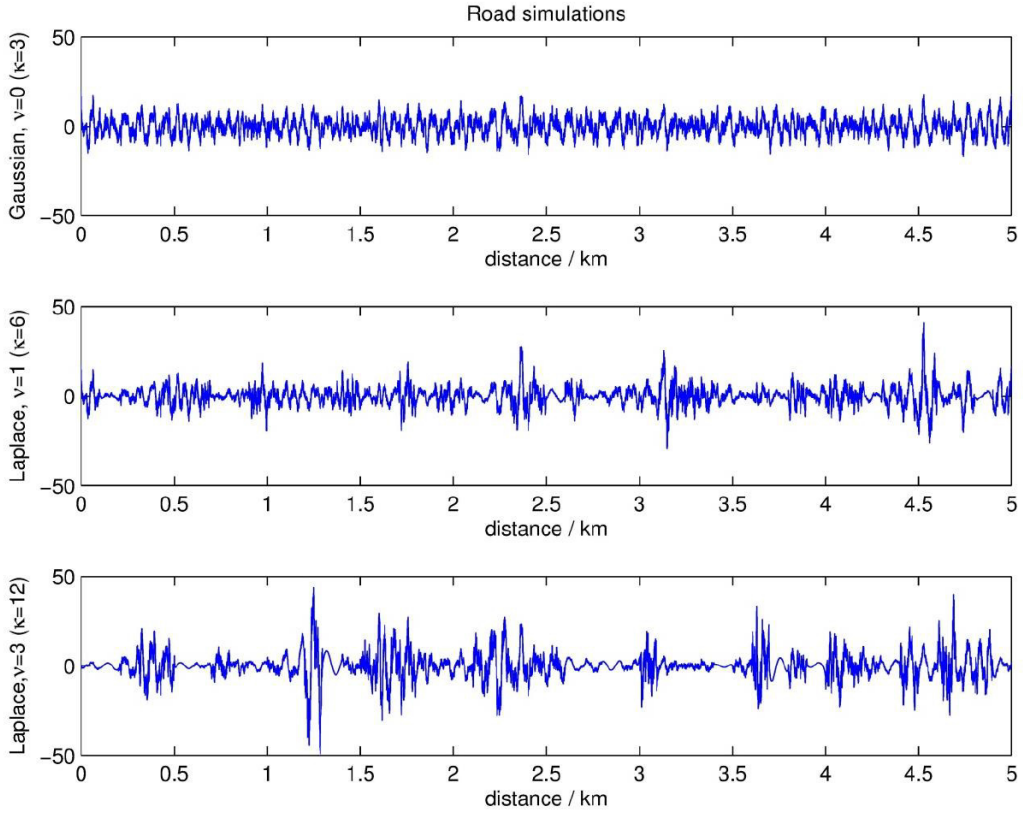
1.2.5 Statistical modeling of road Profiles

As mentioned above (1.2.2) there are well established roughness indicators. As the ISO and IRI are exchangeable in a statistical sense (see 1.1), we will reduce the following descriptions to the ISO 8608 roughness indicator C.

1.2.6 Laplace Model

The roughness index of a road is calculated by using a accumulation statistics on its profile. In the fatigue damage analysis it is of high interest to research ways to go the other way around. Given a roughness index C, what kind of stochastic process can be used to define a realistic and, in terms of fatigue damage analysis, valid road profile. There has been a lot of research on stationary Gaussian processes that are used to accomplish that task. However this approach is not valid for long segments of road and therefore not suitable in simulations that are motivated by real world situations, like in the VMC (1.3) context. Johannesson and Rychlik propose the use use of a piecewise stationary Gaussian process where each piece of the process is given a random standard deviation. In their paper([JR13]) they introduce the idea of a non-stationary Laplace process to model a road's profile. To construct the process they model each 100m by a Gaussian process with mean roughness C and randomly assign a gamma distributed variance defined by the gamma distribution parameter κ resulting in a local variance of ν. In their work they

Figure 1.6: Simulated profiles with different Gamma-distributed variances each 100m segment ([JR13])



validate this model on real life data of the roads in Finland. This yields a two parameter statistical model, which can be used to simulate long roads realistically.
The same authors also introduce a quantity called 'Expected Damage Index' from the Laplace surface model. In their paper [JPR14] they derive a calculation method based on a material constant k, a reference speed v_0 and a reference ISO value C_0

$$E[Dv(k, C, \nu)] \approx 0.07093 e^{13.92k} \left(\frac{C}{C_0}\right)^{k/2} \left(\frac{v}{v_0}\right)^{k/2-1} \frac{\Gamma(k/2 + 1/\nu)}{\Gamma(1/\nu)}$$

1.3 Virtual measurement campaign

This work is motivated by the idea to enrich the roughness information of streets within the environment model of a software tool called the 'Virtual Measurement Campaign' or VMC,

developed by the Fraunhofer ITWM. The VMC software can be used for a variety of tasks within the scope of load data analysis, measurement analysis, region and route analysis and several more types of analysis, using real life and real world environment data. The software relies on a database holding various information used for such analysis. These kinds of data include road related information (roughness, pavement, width,...), their geometric and topographic properties (curvature, hilliness), socioeconomic indicators and much more. In this work we develop a model for the estimation of a target property (in most cases a road roughness indicator) by data of the environment model's road related properties together with the topology of the street networks. Such models can be used to fill gaps of information within the database or to simply gain knowledge about systematic dependencies between the available quantities. By simulating road profiles as in 1.2.5 based on our road roughness estimation we have an instrument for damage estimation of vehicles. This can be done statistically (as in 1.2.5) or by vehicle and driver simulation systems.

1.4 Road statistics

To gain information about the nature of the road roughness and other road related quantities we conducted a statistical study on the street network of Finland. The available data, provided by the Finnish Transport Agency, is described in detail in chapter 7. The dataset contains the measured IRI values for large portions of the Finnish motorways and major roads, alongside with other road related information. The data is assigned to segments of the roads, each segment of length 100m. Since we are targeting for a model that allows extrapolation of roughness information this dataset alone allows for a descriptive analysis. The key figures that we are taking into consideration here are the road roughness, pavement and functional class.

- Road roughness is given in the form of a measured IRI value for each segment.
- The pavement material is one of three classes (asphalt, soft concrete, gravel)
- The road's functional class describes the importance of the road (main road, local road, connection road)

1.4.1 Descriptive statistics

In order to understand the basic characteristics of the dataset we did a study showing descriptive statistics of the data. As road related quantities we chose to look at the surface material and the functional class of the road. A frequency statistic showing the distribution of these quantities is depicted in Figure 1.7. A Gaussian kernel density estimation of the IRI (depicted in Figure 1.8 of the individual segments shows that the underlying marginal distribution may be described by a log-normal distribution. The dependency between the

Figure 1.7: Frequency statistics of the road related quantities

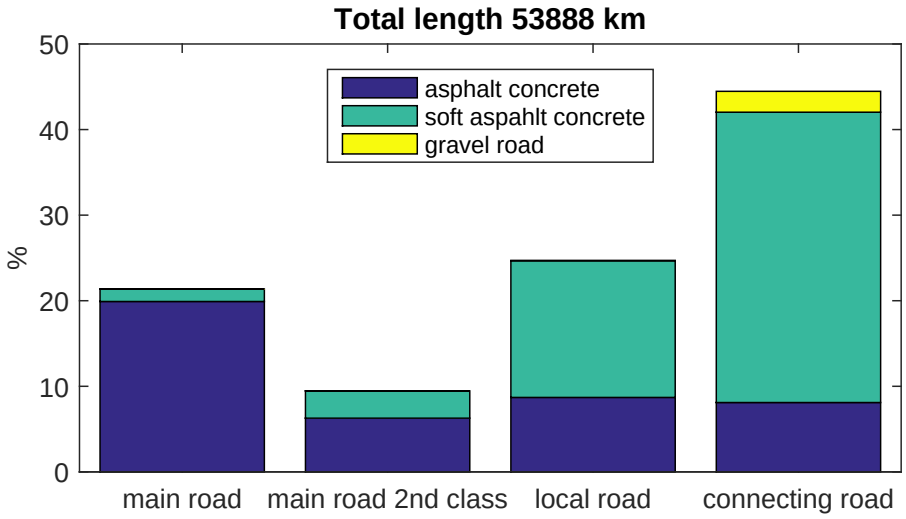
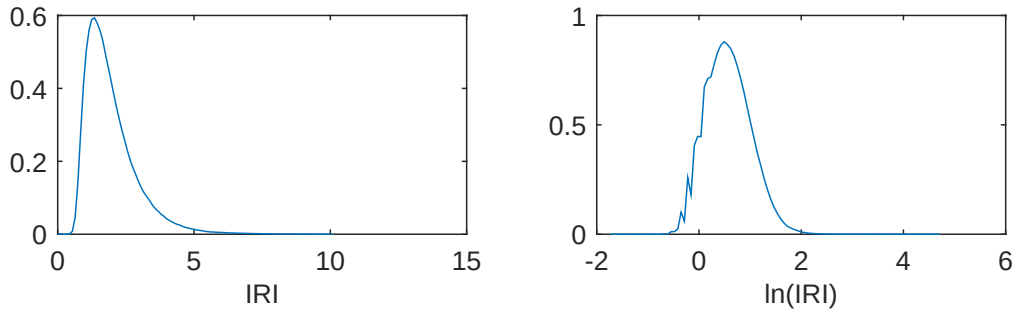


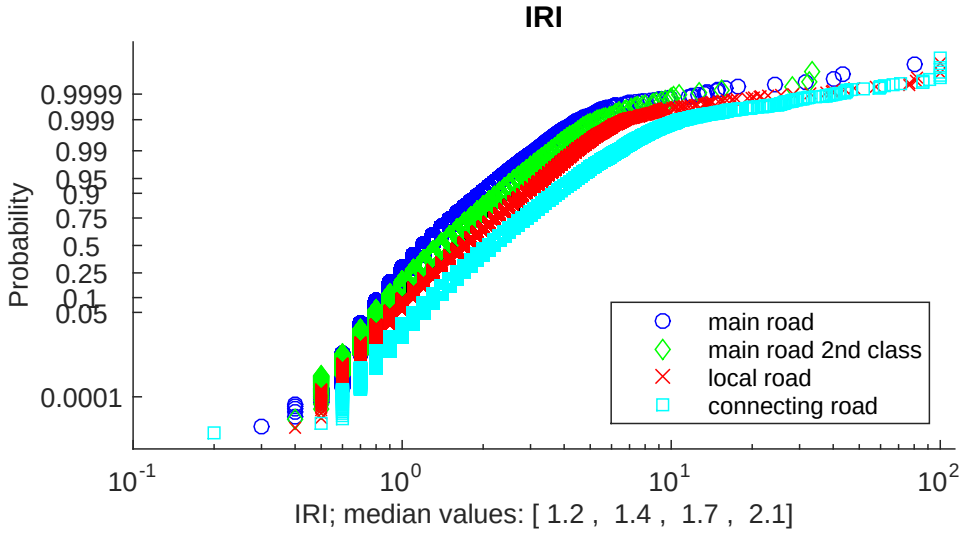
Figure 1.8: Gaussian Kernel Density Estimation of the IRI Distribution. Left the original IRI Data of Finland, right the logarithmic values.



IRI and the road class can be observed in Figure 1.9. The plots are Normal-probability plot of the log-IRI values. We observe clearly distinguishable distributions for the different functional classes as well as for the different pavement materials.

In the Finnish dataset the street segments are geographically ordered in the direction of the streets. This allows us to reconstruct the streets by grouping ordered sequence of variable values for the IRI, functional class and the pavement. This results in a set of street objects of variable length.

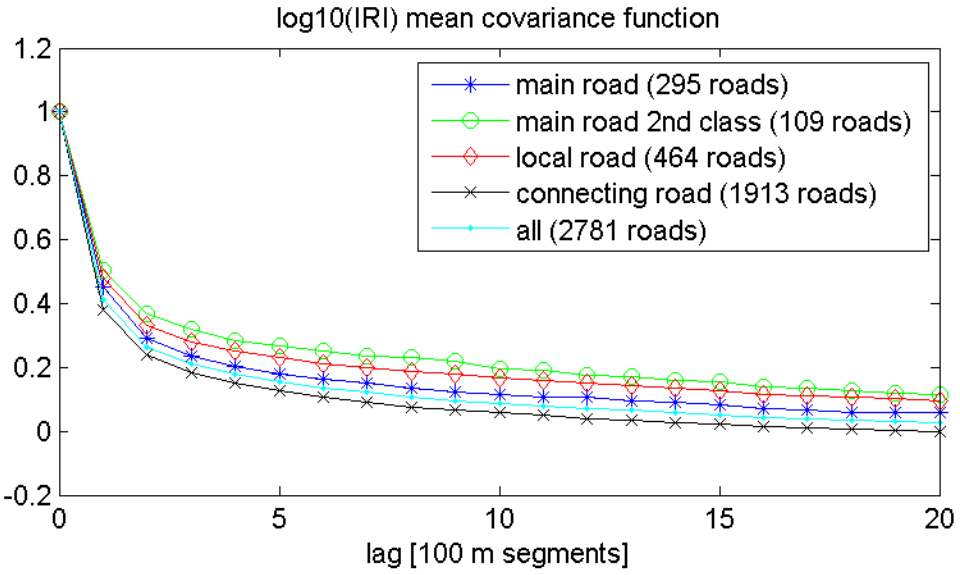
Figure 1.9: Log-Normal probability plot of the IRI grouped by the functional class



1.4.2 Autocorrelation study

The term autocorrelation is used for stochastic processes and therefore we require a series of measurements over an ordered set. We investigate the autocorrelation of the street segments' IRI values when we look at each individual street object as a sequence of IRI values. Therefore the series of IRIs is defined by the connectivity of the roads and the driving direction. Using street segments of all functional classes and pavement types and calculating the mean autocorrelation of the street segments in driving direction, we receive the correlogram depicted in Figure 1.10. In this work, we do not focus on the possibilities of modeling single street's IRI series but on creating a complex model for road roughness in dependency of observable attributes. The strong autocorrelation shown in 1.10 suggests that such a model has to incorporate the neighboring roughness information to a local estimation of the IRI.

Figure 1.10: Correlogram of log-IRI values



1.5 Target of this work

Our goal is to develop a model for discrete figures that can be used for road quality measures like a discretized road roughness index in dependence of observable information like socioeconomic data, surface properties or geometric properties of the road.

The model shall be capable of using the road networks topology since we suspect high dependency of neighboring measurements which has already been statistically shown for the sample of the Finnish road data in the previous section (1.4). Further the model should be trained in a limited amount of time on standard office hardware, for it might be integrated into the workflow of the VMC software.
The structure is:

- Chapter 2 - Introduction of graphical models and reasoning why Conditional Random Fields are a suitable class of models for the extrapolation of discrete road roughness data.
- Chapter 3 - Inference techniques that allow to receive marginal distributions for structures like Conditional Random Fields.
- Chapter 4 - Training of Conditional Random Fields. We focus on supervised learning techniques.
- Chapter 5 - Optimization methods for big graphical models. We focus on stochastic gradient optimization methods in order to be able to train big models with standard office hardware.
- Chapter 6 - The theoretical optimization boundaries are not sharp. We conduct a broad series of studies showing the potential of the different optimization methods on artificial data.
- Chapter 7 - We apply the CRF model to two datasets and compare it to linear modeling.
- Chapter 8 - Conclusion and outlook on possible enhancements as well as selected related work.

2 Modeling

In this chapter we introduce a model that can be used for road quality modeling. We start in section 2.1 by introducing the concept of classification and graphical models separately. In section 2.2 we show different graphical models used for classification and discuss conditional random fields for general graphs in more detail in 2.3. In sections 2.4 and 2.5 we go into more detail in factor modeling and application related topics. We conclude this chapter by defining a model setup for the street quality measurements.

2.1 Classification and graphs

From section 1.4 we conclude that our model should consider the road networks topology for it is a valuable kind of information. This cannot be covered using regular regression models since it is not possible to encode a completely arbitrary graph in dummy variables or other workarounds for linear models. As the road quality is a part of the environment model of VMC we are interested in its probability distribution instead of trying to estimate expected values or defining other point estimators. The distribution allows for add-on simulations of road surfaces. Since our model shall be independent of the type of quality measure we assume that the target quantity can be represented by a random variable with values in a finite set $\mathcal{Y}$. That way we do not make any assumptions on linearity or other intrinsic systematics of the target quantity. Furthermore this way the problem can be regarded as a classification problem. The class of classification models suited for dealing with topological information is called 'Probabilistic Graphical Models' or PGM. A wide range introduction can be found in the work by Koller and Friedman [KF09]. We give a short step by step model introduction to introduce ideas and notations with increasing complexity. This also allows for a comparison of different PGMs and to describe why CRFs are best suited for the classification problem that handles the road quality modeling.

2.1.1 Notation

We are using a very common set of notations as most used in the referenced literature. Here we will give a short summarizing overview to avoid misinterpretation.
Random variables are denoted by capital letters (i.e. X,Y), the corresponding realizations by lowercase letters(x,y). We denote the probability measure of a random variable X by $\mathbb{P}^X$. Since we will often work with conditional probabilities we will stick to that notion. $\mathbb{P}^{Y|X}$ denotes the measure of the random variable $Y|X$. A small p is used to denote a

realizations probability: $p(x) = \mathbb{P}^X(\{x\})$ and $p(y|x) = \mathbb{P}^{Y|X=x}(\{y\})$
Vector valued variables will be written in bold letters (i.e. $\mathbf{X} = (X_1, \ldots, X_n)$). Such variables can be indexed $X_i, i \in \mathbb{N}$. For the application in graphical models we will also allow negative indices $X_{-i} := (X_1, \ldots X_{i-1}, X_{i+1}, \ldots, X_n)$ and set-valued indices $X_C := \{X_i | i \in C\}$. Note that using set valued indices results in a set of random variables.

2.1.2 Classification

Since in our case the target variable Y can be assumed to take on values in a finite set $\mathcal{Y}$ the setup can be regarded as a model for classification. In the classification context we differentiate between observable information, further denoted by random variable $\mathbf{X}$, and non-observable information Y, the target variable. The classification problem is to determine the class label y for an observation vector $\mathbf{x}$. There is a wide variety of algorithms that have been developed in the field of machine learning and computational statistics (e.g. Support Vector Machines [Vap]). Most classification methods use the principle of supervised learning, where a model is trained on a dataset where the observable information $\mathbf{X}$ and labels Y are known ($S_{train} = \{(x_i, y_i)\}_i$). Training is usually done by finding an optimal model parameter $\boldsymbol{\theta} \in \Theta$ for the training dataset S_{train}. The trained model is used to determine the labels on the test dataset $S_{test} = \{x_i\}_i$ of which only the observable information $\mathbf{X}$ is available. As mentioned above, in our case, we are not only interested in the correct labeling but in the probabilistic properties, especially in the conditional distribution $p(Y|\mathbf{X})$. Not all classification algorithms aim for an estimation of such conditional probabilities, since it is not always necessary. Especially in the field of image processing and the distribution itself is not to be determined but only the most likely labeling. For such scenarios the above mentioned SVMs are sufficient. However once you have calculated an estimation of the probabilities $\hat{p}(Y|\mathbf{X})$, the expected value $\mathbf{E}_{\hat{p}}[Y|\mathbf{x}]$ can be used for classification. Models that directly target for estimations of the conditional probability are called discriminative, while models aiming to estimate the joined distribution $p(\mathbf{X}, Y)$ are called generative. Since in the classification scenario the term $p(\mathbf{X})$ is constant, the classifier can be based on the joined as well as on the conditional probability estimator:

$$p(y|x) \propto p(y, x)$$
$$\hat{y} = \text{argmax}_y \hat{p}(y|x) = \text{argmax}_y \hat{p}(y, x)$$

In the next subsections we will briefly discuss the Naive Bayes Classifier and the Maximum Entropy Classifier which deal with classification problems without topology consideration. The graphical models follow afterwards.

2.1.3 Naive Bayes

The naive Bayes is a basic generative classifier. It models a single scalar label Y and a vector of observations $\mathbf{X}$. The main idea is to simplify the estimation of the joined

probability of label and observation by assuming conditional independence between the observations given the label information.

$$\mathbb{P}(\mathbf{X}|Y) = \prod_{i=1}^{n} \mathbb{P}(X_i|Y)$$

That way the joined probability becomes:

$$p(y, \mathbf{x}) = p(y) \prod_{i=1}^{n} p(x_i|y)$$

The observations are usually discrete or of an assumed parametric distribution. For discrete observations and a given training set $\{(x_i, y_i)\}_{i \in S}$ a standard approach for the estimators is:

$$\hat{p}(x_i = k|y = b) = \frac{\#_S\{x_i = k, y = b\} + l}{\#_S\{y = b\} + J_i l}$$

where J_i is the number of discrete values that x_i can take on and l is a free smoothing parameter. This can either be regarded as a smoothed maximum likelihood estimator or as a Maximum-A-Posteriori (MAP) estimator with Dirichlet priors ([Mit97]). The label estimator $\hat{p}(y = b)$ can be defined similarly. For continuous observations one often assumes Gaussian or other low-parametric distributions and uses the MAP estimators the same way as in the discrete case. The resulting generative classifier can be defined as follows:

$$l_{\text{Gen}} = \text{argmax}_j \frac{\prod_i \hat{p}(x_i|y = y_j)\hat{p}(y = y_j)}{\sum_{y'} \prod_i \hat{p}(x_i|y = y_j)\hat{p}(y = y')}$$

An essential weakness of the naive Bayes classifier is its strong conditional independence assumption. It assumes that all information about the interaction of observations can be sufficiently encoded in the class label. Mitchell ([Mit97]) gives a nice example where such an assumption is very reasonable, in which 'Rain' is independent of 'Thunder' conditioned under 'Lightning'. This example demonstrates that the independence assumption can be reasonable but on the other hand it shows how strong the implication is and that the assumption should be used in cases where an interaction of observations is unknown.

2.1.4 Maximum entropy classifier

Another way to define a discriminative classifier is by using multivariate logistic regression. The conditional probabilities are defined as such:

$$p(y|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \exp\left(\theta_{y,0} + \sum_{j=1}^{J} \theta_{y,j} x_j\right)$$

where $Z(\mathbf{x}) := \sum_{y' \in \mathcal{Y}} \exp\left(\theta_{y,0} + \sum_{j=1}^{J} \theta_{y',j} x_j\right)$ is the normalization function, $\boldsymbol{\theta}_y$ is the model parameter specified for each $y \in \mathcal{Y}$ and J denotes the size of the parameter space. Before going into more detail we want to introduce a slightly different way of writing the conditional probability definition. Following a commonly used notation (e.g. in [SM10]) let

$$f_{i,j}(\mathbf{x}, y) := \mathbb{1}\{y = y_i\} x_j \quad \text{with} \quad f_{i,0}(\mathbf{x}, y) := \mathbb{1}\{y = y_i\}$$

The set of feature functions $\{f_{i,j}\}_{i,j}$ is finite and can be reordered to be written as a vector $\{f_{i,j}\}_{i,j} = \{f_k\}_k$ to fit the formulation:

$$p(y|x) = \frac{1}{Z(\mathbf{x})} \exp\left(\sum_{k=1}^{K} \theta_j f_k(\mathbf{x}, y)\right) = \frac{1}{Z(\mathbf{x})} \exp\left(\boldsymbol{\theta}' \mathbf{f}(\mathbf{x}, y)\right)$$

The normalization or partition function $Z(\mathbf{x})$ is the first notable difference between naive Bayes and the logistic regression. This term can be computationally expensive for big models with $|\mathcal{Y}|$ being of large magnitude. Such a normalization is needed for all classification approaches that do not, like Bayes, assume that the distribution for X factorizes into marginals (with or without conditioning). Further the logistic regression defines a model that prevents overfitting quite naturally. In a family of certain 'suitable' conditional probabilities the above definition of $p(y|\mathbf{x})$ maximizes the entropy. The resulting classifier is hence called Maximum Entropy Classifier. In more detail this means that given a training set $(\mathbf{x}_i, y_i)_i$ the logistic regression conditional distribution is the solution to the following constraint optimization problem:
Find a probability distribution π with maximal (observed) entropy:

$$\pi = \text{argmax} H(p) \quad \text{with} \quad H(p) := -\sum_{y' \in \mathcal{Y}} \sum_{x \in M} p(Y = y'|x_i) \log\left(p(Y = y'|x_i)\right)$$

under the constraints

$$\forall k \leq K, y' \in \mathcal{Y}: \qquad \sum_i p(y'|x_{j,i}) x_{j,i} = \sum_i f(\mathbf{x}_i, y_i)$$

The idea behind maximum entropy as optimization criterion, is that the distribution is as uniform as possible given the above constraints. This yields a distribution that rules out events of zero probability. Further it ensures that invalid or missing observations can be incorporated by setting $f(\mathbf{x}_i, y_i) = 0$ for all unobserved or invalid pairs $(\mathbf{x}, y)_i$.

2.1.5 Graphical models

Probabilistic graphical models use graph structures to represent random variables and their stochastic dependencies. In general we can distinguish between directed and undirected graphical models. A graph is defined by a set of vertices $V = \{1, \ldots, n\}$ and a set of adjacency relations or edges E. Hence we denote a graph by $G = (V, E)$.

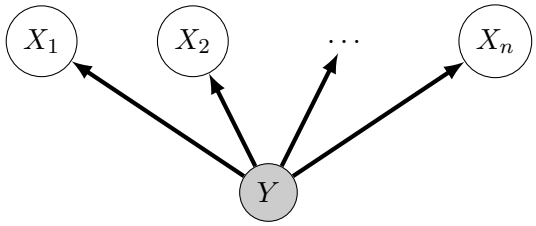
Definition 2.1.1 (Directed graphical model)**.** *A multivariate random variable* $\mathbf{X}$ *of dimension* $n = |V|$ *is called a directed graphical model corresponding to graph* $G = (V, E)$ *if and only if:*

$$\mathbb{P}(\mathbf{X}) = \prod_{i=1}^{n} \mathbb{P}(X_i | X_{\pi(i)}) \quad \textit{with} \quad \pi(i) = \{j \in V | (j, i) \in E\}$$

where $\pi(i)$ *denotes the set of parent nodes for each node* i*. If the graph* G *is a directed acyclic graph (DAG) the model is called a Bayesian network.*

The model behind the naive Bayes classifier is an example for a directed graphical model. Since there are no cycles it is also a Bayesian network. The graphical structure is given by the relation $(Y, X_i) \in E$ for $i \leq n$. A graphical interpretation is given in figure 2.1.

Figure 2.1: Graphical model for naive Bayes classifier



Definition 2.1.2 (Factor, Undirected graphical model)**.** *Given a multivariate random variable* $\mathbf{X}$ *and an index set* A*, a non negative function, depending only on* $\mathbf{X}_A$ *is called a factor. Notation is* $\Psi_A(\mathbf{x}_A)$*.*
A multivariate random variable $\mathbf{X}$ *of dimension* $n = |V|$ *is called an undirected graphical model if*

$$p(\mathbf{x}) = \frac{1}{Z} \prod_{A \in \mathcal{A}} \Psi_A(\mathbf{x}_A)$$

for a given set of factors $\mathcal{A}$ *and a normalizing constant* Z*. If the factors correspond to the (maximal) cliques of* G*, we say that* $\mathbf{X}$ *factorizes according to* G*. The distribution* $\mathbb{P}(X)$ *is then called Gibbs distribution.*

Definition 2.1.3. *An undirected graphical model* $(\mathbf{X}, G = (V, E))$ *is called a Markov random field (MRF) iff* $\forall X_v$

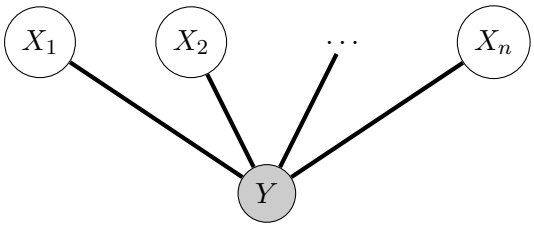
$$\mathbb{P}(X_v | X_{V \setminus v}) = \mathbb{P}(X_v | X_{N_v})$$

where $N_v := \{w \in V | (w, v) \in E\}$ *denotes the set of neighbours of a vertex* i*.*

Theorem 2.1.4 (Hammersley-Clifford)**.** *An undirected graphical model* $(\mathbf{X}, G)$ *factorizes according to* G *iff it is a Markov Random Field.*

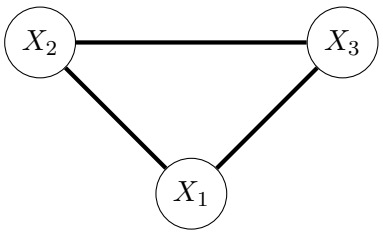
The original paper containing theorem 2.1.4 is an unpublished work by Hammersley and Clifford [HC71]. The important aspect of the Hammersley-Clifford theorem is that it states that any Gibbs distribution on an undirected graphical model can be defined via the marginals conditioned under the neighborhoods. This allows us to transfer the idea of the logistic regression to graph based classification, by defining its local conditional distributions. Although it is sufficient to specify a Gibbs distribution by defining potentials

Figure 2.2: Graphical model for logistic regression



on the maximal cliques it is often more plausible to use smaller cliques. However using the graphical notions from above we cannot graphically distinguish between an undirected model with all potentials defined on maximal cliques and a Markov random field with potentials on smaller cliques. A simple example can be found in [SM10]. It is depicted in figure 2.3. This graph represents a Markov network. However the factorization into

Figure 2.3: Markov network with ambiguous factorization



potentials is not uniquely defined by the graph. Possible factorizations are:

$$p(\mathbf{x}) = \Psi_1(X_1, X_2) \cdot \Psi_2(X_2, X_3) \cdot \Psi_3(X_3, X_1)$$

or

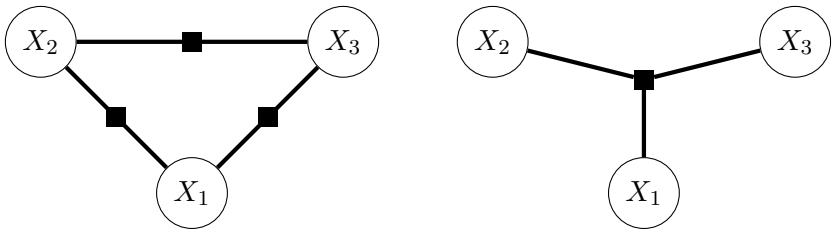
$$p(\mathbf{x}) = \Psi(X_1, X_2, X_3)$$

In order for the graphical structure to hold all information about the factorization we use factor graphs.

Definition 2.1.5 (Factor graph)**.** *A factor graph is a biparted graph $G = (V \cup \mathcal{F}, E)$ with the following partitions. V is the set of nodes corresponding to a random field $\mathbf{X}$ and $\mathcal{F}$ a set of nodes corresponding to potentials. A variable $X_v \in V$ and a factor $\Psi_A \in \mathcal{F}$ are connected iff X_v is an argument for Ψ_A.*

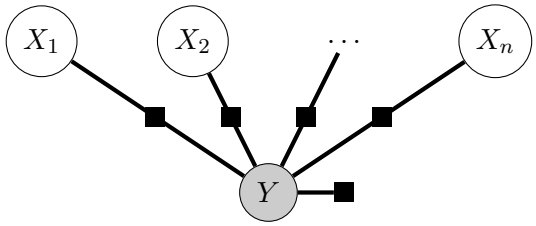
We represent factor nodes by black squares when depicting graphs. The ambiguous Markov factorization from the above example of [SM10] depicted in figure 2.3 can therefore be more precisely formulated by one of the graphs in figure 2.4.

Figure 2.4: Factor graph representations



Using the factor graph notation we can depict the Maximum Entropy classifier as seen in figure 2.5.

Figure 2.5: Graphical model for logistic regression model



2.2 Classification on graphical models

In this section we extend the classification methods above to multivariate labeling problems. We now assume $\mathbf{Y}$ to be a random field of labels from the same value set $\mathcal{Y}$. Therefore $\mathbf{y} \in \mathcal{Y}^{|V|}$ where V is a finite set. The same set also serves as primary index for the observations. In graph based classification we denote them by $\mathbf{X} = (\mathbf{X}_v | v \in V)$, where each $\mathbf{X}_v$ is a multivariate random variable $\mathbf{X}_v = (X_{v,1}, \ldots, X_{v,m})$ where m is the amount of different observations for a given vertex. This number is assumed to be fixed $\forall v \in V$. A graphical model is defined as in the previous section with the set of vertices being $V_{\mathrm{cl}} = \{Y_v, \mathbf{X}_v | v \in V\}$.

Performing classification on a graph $G = (V, E)$ means classifying the labels via the observed information on each vertex $v \in V$. This assumes that for each vertex $v \in V$ we can define a label representation $Y_v \in \mathcal{Y}$ and a random variable representing the observation $\mathbf{X}_v$. One major goal in classification is to find a model which describes the

probability of a label realization $\mathbf{y}$ given an observation $\mathbf{x}$ for the complete graph.

$$p(\mathbf{y}|\mathbf{x}) \quad \text{with} \quad \mathbf{y} = \{y_v\}_{v \in V} \quad \text{and} \quad \mathbf{x} = \{\mathbf{x}_v\}_{v \in V}$$

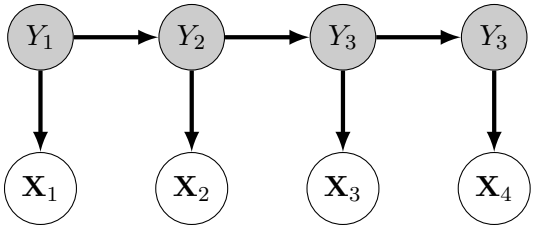
This is not equivalent to classifying each vertex individually, but allows for modeling complex interactions of labels and observations.
In the following subsection we will assume the underlying graph to be linear, as it is the simplest non-trivial graphical structure. There are many applications in the field of natural language processing where such a simple graphical structure is sufficient. A single street partitioned by a set of measurement locations can also be modeled with such a simple graph. Most important however is that we want to compare different models in terms of generality, performance and further criteria. This can already and most clearly be demonstrated on linear graphs.

2.2.1 Hidden Markov model

Linear hidden Markov models are very popular in the field of natural language processing or named entity recognition (e.g. [Rab89]). The model is generative and makes two assumptions for the joined probability of $\mathbf{X}$ and $\mathbf{Y}$:

1. Markov property $\mathbf{Y}$: $p(\mathbf{y}, \mathbf{x}) = \prod_{t=1}^{V} p(y_t|y_{t-1})p(\mathbf{x}_t|y_t)$
2. Naive Bayes: $p(\mathbf{y}, \mathbf{x}) = \prod_{t=1}^{T} p(y_t|y_{t-1}) \prod_{i=1}^{n} p(x_{t,i}|y_t)$

Figure 2.6: Linear hidden Markov model (HMM)



HMM have been used with great success fields where these assumptions are suitable. Usually this goes along with the graph representing a time associated entity as this makes a directed model plausible. Especially problems with very big observation ranges are usually well suited for HMM. In the language oriented problems, the X_i often represent words from a huge dictionary. In a natural language the independence assumption of neighboring words is not true but leads to small errors. The fewer words there are in the dictionary the bigger the relative error that is going along with the naive Bayes assumption.
A lot of work has been put into training and inference of HMMs. While supervised learning is usually performed by maximum likelihood estimation, there are several methods for effective unsupervised training as well (i.e. [Sha11]).

2.2.2 Maximum-entropy Markov model (MEMM)

The maximum entropy Markov model, proposed by McCallum, Freitag and Pereira [MFP00], is of a similar structure as the HMM. However it does not model the distribution of the observations $\mathbf{X}$ and is therefore a discriminative approach. The idea is to split the conditional distribution at a specific node into a state transition probability $\mathbb{P}(y|y')$ and a set of $|\mathcal{Y}|$ label specific probabilities conditioned under the observations.

$$p(y, y^-, \mathbf{x}) = p_{y^-}(y|\mathbf{x}) = \frac{1}{Z(\mathbf{x}, y^-)} \exp\left(\boldsymbol{\theta}' \mathbf{f}(y, \mathbf{x})\right)$$

with $Z(\mathbf{x}_t, y^-)$ being the partition function that ensures $\sum_y p_{y^-}(y) = 1$. The notation $y^- \in \mathcal{Y}$ is chosen to indicate that it is an arbitrary value in $\mathcal{Y}$ but in practice usually the predecessor value of the vertex in question. The parameter vector $\boldsymbol{\theta}$ and the feature function vector $\mathbf{f}$ are defined exactly as in the maximum entropy classifier setup 2.1.4. It also ensures maximum entropy under the constraint:

$$\frac{1}{m_{y^-}} \sum_{k=1}^{m_{y^-}} \mathbf{f}\left(\mathbf{x}_{t_k}, y_{t_k}\right) = \frac{1}{m_{y^-}} \sum_{k=1}^{m_{y^-}} \sum_{y'} p_{y^-}(y'|\mathbf{x}_{t_k}) \mathbf{f}\left(\mathbf{x}_{t_k}, y'\right)$$

with the set of nodes $t_1, \ldots, t_k$ being the nodes where $y_{t_k} = y^-$. Like in HMM the dependency of the label variables is assumed to be form a directed graphical model with the corresponding Markovian structure.

$$p(\mathbf{y}, \mathbf{x}) = \prod_t p(y|y_{t-1}, x)$$

This is the general MEMM definition according to the initial paper by McCallum, Freitag and Pereira [MFP00]. The Markov property implies the conditional Gibbs probability definition which in the linear context can be written as:

$$\mathbb{P}(y_t|y_{t-1}, \mathbf{x}_t) = P_{y_{t-1}}(y_t|\mathbf{x}_t) = \frac{1}{Z(y_t, y_{t-1})} \exp\left(\boldsymbol{\theta}' \mathbf{f}(y_t, y_{t-1}, \mathbf{x}_t)\right)$$

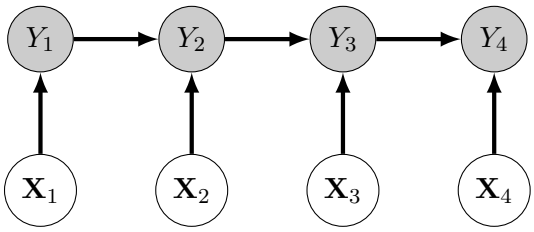
This way of parametrizing $P_{y^-}(y|\mathbf{x})$ on the one hand leads to a wide variety of possible training algorithms as well as plausible feature definitions $\mathbf{f}$ and inference algorithms. For most algorithms designed for Hidden Markov Models there are variants for the training and inference of MEMM.

2.2.3 Linear Conditional Random Fields

The last basic linear model we will introduce in this context is another way to extend the maximum entropy classifier to graphical classification problems. Unlike MEMM, the basis for conditional random fields is an undirected graphical model with cliques $a \in \mathcal{A}$ are beeing the tuples $(t, t+1)$:

$$\mathbb{P}(\mathbf{y}|\mathbf{x}) = \prod_A \Psi_A(\mathbf{y}_A, \mathbf{x}_A) = \prod_t \Psi_t(y_t, y_{t-1}, \mathbf{x}_t)$$

Figure 2.7: Linear maximum entropy model (MEMM)



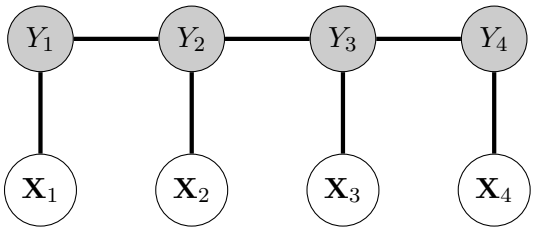
The conditional distribution does not factorize into local probabilities as it does in directed graphs. Like in the logistic regression classifier the Hammersley Clifford theorem (2.1.4) states that the Markov property is equivalent to the gibbs distribution which can be written as:

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{t=1}^{T} \exp\left(\sum_{k=1}^{K} \theta_k f_k(y_t, y_{t-1}, \mathbf{x}_t)\right) = \frac{1}{Z(\mathbf{x})} \prod_{t=1}^{T} \exp\left(\boldsymbol{\theta}' \mathbf{f}(y_t, y_{t-1}, \mathbf{x}_t)\right)$$

Notice that this form is very similar to the MEMM conditional distribution. However due to the factorization of the probability into local probabilities the partition function $Z(\cdot)$ of the MEMM is locally defined, while it is the sum of all random field states in the case of CRF.

$$Z(\mathbf{x}) = \sum_{\mathbf{y}'} \prod_{t} \exp\left(\boldsymbol{\theta}' f(y'_t, y'_{t-1}, \mathbf{x}_t)\right)$$

Figure 2.8: Linear conditional random field



2.2.4 Model comparison

The above models have been compared in many applications and in many variants and applications (i.e. [SM10]). Their generalizations to general graphs are to a certain degree suitable for modeling a road network. We will make a short comparison in the main aspects that are relevant for the road network modeling.

Generative vs. discriminative

The main difference between generative and discriminative models is the consideration of the distribution of the observation $\mathbb{P}^{\mathbf{X}}$. Generative models use the joined probability $\mathbb{P}^{\mathbf{Y},\mathbf{X}}$ for classification and therefore implicit or explicit knowledge about the distribution of $\mathbf{X}$. This can be beneficial if their is profound knowledge about $\mathbf{X}$ or at least a justifiable assumption. To keep computational costs limited generative models like HMM make rather strong independency assumptions on the nature of the observations. If these assumptions are justified a generative model can be reasonable to use. In general it is a hard problem to use generative models with overlapping observations. Then either the simplifying assumptions are unjustified or it is hard to estimate the correct joined distribution.
Ng and Jordan discuss in their a paper [NJ01] generative and discriminative classifiers. They show that while the training convergence of generative models is much faster, there is an asymptotic bias. This results from falsely assumed stochastic independence and further effects. Discriminative models have no such bias, but converge much slower.
Since the street quality model is to be used in a user controlled dynamic environment (VMC software 1.3)with possibly large number of observations and also overlapping features we cannot justify conditional independence of the observations.
In order to use models like HMM preprocessing of the observation is needed to obtain a situation with an independent observation vector $\mathbf{X}$. This task is also not applicable in general since the observations can not be modeled in a single distribution family. Therefore different sets of transformations have to be applied for each dataset. The above problems only occur due to the modeling of the distribution of $\mathbf{X}$ and do therefore not occur in CRFs or MEMMs.
A drawback of discriminative models is that information about the distribution of $\mathbf{X}$ cannot be used directly, even if available. In our application however this kind of information is usually unavailable. Another drawback is the higher computational complexity. In MEMMs as well as CRFs the calculation of a partition function Z is needed which can be computationally expensive.

Directed or undirected

We will be associating the nodes of a graph with measurement locations. Therefore the label of a vertex is directly corresponding to the quality measurement at this location. Intuitively one would expect the directional model to be sufficient whenever their is a single designated driving direction. As true such an intuition is for natural language tasks, where the words of a sentence are associated with the time, it is not justifiable for the street quality situation, which has a spatial dependency (as discussed in 1.4). Another, plainly mathematical reason to not use MEMM or other locally partitioned models is the so called label bias problem [Laf01]. It states that the model is biased towards states with few successors or transition probabilities of low entropy. Such directed models suffer from the fact that a label at a given node is conditionally independent of observations of higher index t. This is usually not true and especially problematic for an extension of the model to non-linear graphical models.

Model choice

For all three linear model classes (HMM, MEMM, CRF) there are variants for general graphical models, that share the strengths and weaknesses with their respective linear variants. Therefore the complexity of the graph does not favor or disqualify one of these models. Basically there are two major aspects we have to consider when choosing a model for the road quality. On the one hand the model has to have as little systematic error and allow a maximum of design freedom since it is user defined and we have no control over the quality of the data or the observations distribution. On the other hand it has to be computationally inexpensive since it is supposed to be calculated on standard PC hardware in acceptable runtime.
HMM has strong assumptions on the observation distribution and is therefore not well suited for our application. MEMM as discriminative approach does not suffer from high assumptions but limits itself by the factorization into local probabilities which results in the label bias problem. We decided to use conditional random fields for it has but few assumptions on the data and has a high degree of freedom in designing the features. Since it major drawback is the computational complexity we focus in this work on the efficient training and inference of such models.

2.3 Conditional Random Field Setup

As mentioned above there are variants for general graphs for all the introduced models. We will however focus on CRFs in the following sections.

2.3.1 General CRFs

Let $\mathbf{C}$ denote a set of cliques, and therefore subsets of V. For every element $c \in \mathbf{C}$ we define $\mathbf{y}_c := \{y_v | v \in c\}$. A general definition of a conditional random field is now given by

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{c \in \mathbf{C}} \exp\left(\sum_{k=1}^{K} \theta_k f_k(\mathbf{y}_c, \mathbf{x}) \right) = \frac{1}{Z(\mathbf{x})} \prod_{c \in \mathbf{C}} \exp\left(\boldsymbol{\theta}' \mathbf{f}(\mathbf{y}_c, \mathbf{x}_c) \right)$$

with partition function $Z(\mathbf{x})$ defined by:

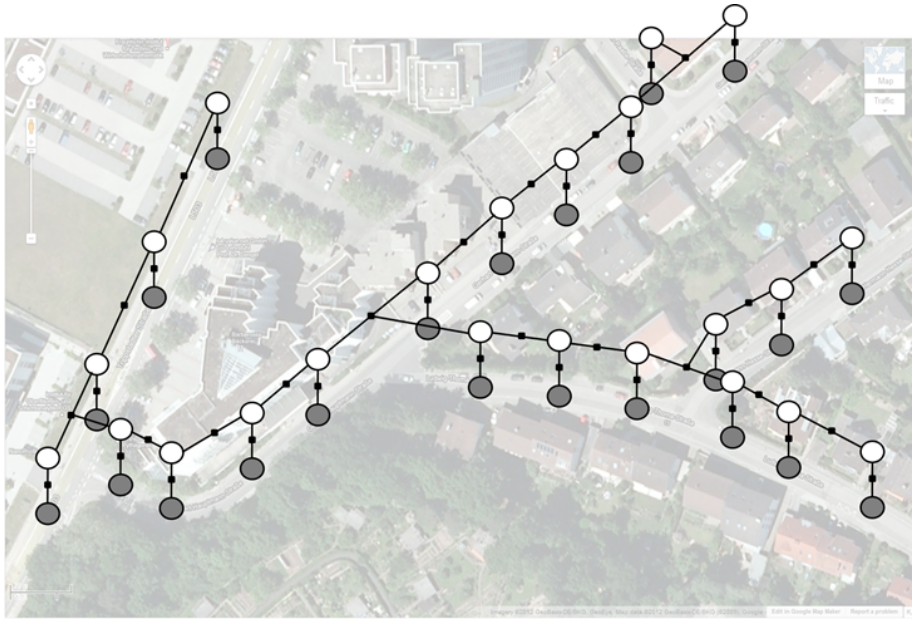
$$Z := \sum_{\mathbf{y}' \in \mathcal{Y}^V} \prod_{c \in \mathbf{C}} \exp(\boldsymbol{\theta}' \mathbf{f}(\mathbf{y}', \mathbf{x}_c))$$

Unlike in the models we introduced so far, in a general CRF we do not limit the function class for the feature functions to indicator functions. Therefore we just assume f_k to be an arbitrary function with values in $\mathbb{R}$.

The main aspect of this work is to define and discuss the parameters of the model and to find ways to efficiently train and apply such a model. While the next two chapters

will discuss efficient training and inference of CRFs, in this chapter we will define the application specific model parameters. Essentially this is achieved by defining a suitable set of cliques $\mathbf{C}$ and a set of feature functions $\mathbf{f}$. Before we start these definitions we will deal with an application specific difficulty, which requires us to use a variation the general CRF definition above.

Figure 2.9: General CRFs used for road roughness measurements

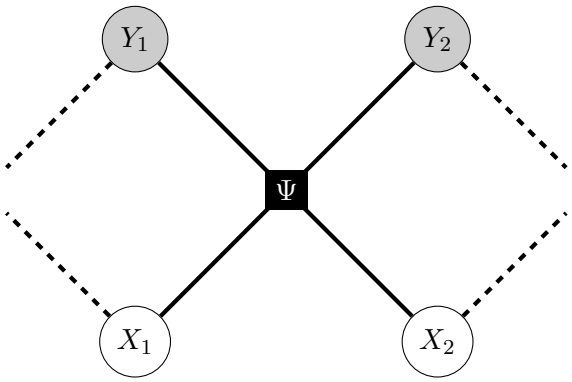


2.3.2 Template based modeling

We are looking for a model that allows us to model systematic dependencies of road related quantities and the road network's topology. Using classification algorithms in that scenario differs from classical tasks in image or language processing in a key aspect. We are usually dealing with a single connected dataset instead of a set of samples that can be regarded as being independent. Sutton and McCallum [SM10] introduce the notion of clique templates to cope with reoccurring clique configurations within a model. These clique templates are representations for a set of cliques on the original graph and are used to partition the set of factors of the model. Since the factors Ψ_A are defined on the cliques of a graph they can now be defined on clique templates.

$$\mathcal{C} = \{C_i\}_i \quad \text{with} \quad \bigcup_i C_i = F$$

Figure 2.10: Template Design



The conditional distribution can be written as:

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(x)} \prod_{C_p \in C} \prod_{\Psi_c \in C_p} \Psi_c(\mathbf{x}_c, \mathbf{y}_c; \boldsymbol{\theta}_p) \quad \text{with} \quad Z(x) := \sum_{\mathbf{y}'} \prod_{C_p \in C} \prod_{\Psi_c \in C_p} \Psi_c(\mathbf{x}_c, \mathbf{y}_c{}'; \boldsymbol{\theta}_p)$$

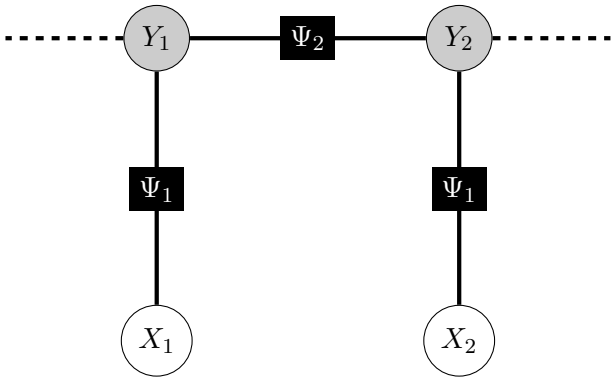
The potentials are defined for every factor c:

$$\Psi_c(\mathbf{x}_c, \mathbf{y}_c; \boldsymbol{\theta}_p) = \exp\left(\boldsymbol{\theta}_p' \mathbf{f}_p(\mathbf{x}_c, \mathbf{y}'_c)\right)$$

Since clique templates group together cliques with identical model parameters it might be useful to identify such groups within our use case as a preprocessing step before the training of the model. However we decided to use a single template for the complete dataset instead of partitioning for example different groups like junctions, street-type based templates or other possible partitions. This decision does effectively not limit the power of the model, since a multiple template setup can be simulated by a single template and a template-specific dummy variable as observation. Therefore it is equivalent to define a set of clique templates and to use a single template with a dummy variable. It appears more natural not to predefine a set of templates since the training of the model and especially model selection algorithms are better suited for our application. However this decision also implies that we are using minimal cliques for the templates, which means that the domain of a potential are the nodes of a single edge. The conditional distribution is now given by:

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{e \in E} \Psi(\mathbf{y}_e, \mathbf{x}_e, \boldsymbol{\theta}) = \frac{1}{Z(\mathbf{x})} \prod_{e \in E} \exp\left(\boldsymbol{\theta}' \mathbf{f}(\mathbf{y}_e, \mathbf{x}_e)\right) \tag{2.1}$$

Figure 2.11: Vertex- and edge-features



2.4 Factor modeling

2.4.1 Potentials

The above definition (2.1)of a single template CRF on general graphs can be specified in more detail to enhance the model. This is done by factorizing the potential into sub-potentials. One possibility to define more specific potentials is to assume that the clique potential can be split up a 'local' potential and an 'network' potential. A local potential models the compatibility of a label and an observation at a single vertex which resembles a single location in most applications. 'Network' potentials are used to model the dependencies between two neighboring label nodes.

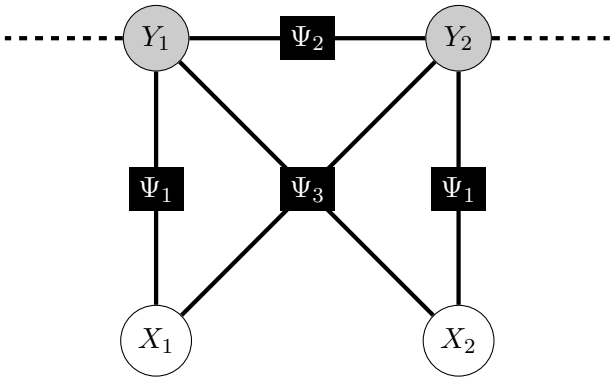
$$\text{Local potential}: \quad \Psi_1(y_v, \mathbf{x}_v) \tag{2.2}$$
$$\text{Network potential}: \quad \Psi_2(y_v, y_w) \tag{2.3}$$

This respective factor graph is depicted in figure 2.11. This potential definition is very intuitive and is similar to the strict separation of edge- and vertex features of linear MEMM (fig: 2.7). In section 2.3.2 we mentioned that the single templated CRF does not limit the general CRF model due to the possible use of dummy variables. However defining the potentials as in 2.2 reduces the use of such methods. For example in practice, a common request is to use a different network potential for different classes of roads. This is not possible to model with the above potentials. Therefore we extend the potential definition by also allowing the most general potential definition possible on the cliques.

$$\text{Local potential}: \quad \Psi_1(y_v, \mathbf{x}_v) \tag{2.4}$$
$$\text{Network potential}: \quad \Psi_2(y_v, y_w) \tag{2.5}$$
$$\text{Clique potential}: \quad \Psi_3(y_v, y_w, \mathbf{x}_v, \mathbf{x}_w) \tag{2.6}$$
$$\tag{2.7}$$

The corresponding factor graph is depicted in 2.12. The Hammersley Clifford theorem

Figure 2.12: Vertex-,edge- and crossing-features



[HC71] tells us that all of the above potentials for CRFs can always be written in an exponential form.

$$\Psi_i(\cdot) := \exp(\boldsymbol{\theta}_i' \mathbf{f}_i(\cdot))$$

The definition of the functions $\mathbf{f}$ is called feature modeling and since in definition 2.3 we place no restrictions on these functions, choosing the set of functions carefully can speed up the model training as well as lead to more plausible results.

2.4.2 Feature modelling

Feature functions play a key role when using conditional random fields. They can be viewed as a measure for the compatibility of a label and a vector of observations. A high feature function value represents a high compatibility of the variable realizations. A trained model uses weights for these compatibilities to define the overall distribution. In this section we will discuss different strategies for defining a suitable set of functions that serve as feature functions for conditional random fields. In general this set of functions is only limited by the type of information the label $\mathbf{y}$ and the observations $\mathbf{x}$ carry as well as the computation time for inference and training.

Discrete indicators

Considering the case of a univariate label y and observation vector $\mathbf{x}$ we can define a set of feature functions as

$$f_{\mathbf{x}',y'}(x, y) := \mathbb{1}\{\mathbf{x} = \mathbf{x}'\}\mathbb{1}\{y = y'\} \quad \text{with} \quad \mathbf{x}' \in \mathcal{X} \quad \text{and} \quad \mathbf{y}' \in \mathcal{Y}$$

Indexed over $\tilde{x} \in \mathcal{X}$ and $\tilde{y} \in \mathcal{Y}$ the amount of different feature functions is finite iff $\mathcal{X} < \infty$ and $\mathcal{Y} < \infty$, which makes the set of all feature functions defined in such a way

$$\mathcal{F}_1 = \{f_{\mathbf{x}',y'} | \mathbf{x}' \in \mathcal{X}, y' \in \mathcal{Y}\}$$

However in many applications the curse of dimensionality leads to an intractable amount of feature functions. Furthermore this approach is limited to variables over finite spaces and therefore not applicable to all data scenarios.
This feature definition is mainly used for 'local' potentials. However it is applicable for all kinds of potentials that have been described above. However using indicator function sets for clique potentials is highly unlikely since the number of functions squares when defining them for all clique configurations. Let $r_i \in \mathbb{N}_+$ be the number of different realizations of $X_{v,i}$ for any vertex $v \in V$. Then the total number of different feature functions that can be defined by discrete indicators as above is:

$$|\mathcal{F}_1| = |\mathcal{Y}| \cdot \prod_i r_i$$

Approximating non-discrete observations

Dropping the restriction of finite spaces for the observations, we define another set of feature functions. In order to account for a wide variety of compatibilities of labels and observations we only assume that the contribution of x is of continuous nature. $f_{\tilde{y},g}(y, \mathbf{x}) = \mathbb{1}\{\tilde{y} = y\}g(\mathbf{x})$ with a continuous function $g(\cdot)$. This can be seen as a generalization of the naive combinations from the previous section. In order to receive a finite set of functions we have to further simplify the function class of g to a parametric set of functions. A suitable approach is to set a maximum approximation level N and approximate the continuous function g by a polynomial of degree N in a given base. We will start by defining the functions on scalar observations. With a monomial base we can define the approximation function by:

$$\tilde{g}_{\theta_1,\dots,\theta_N}(y, x) := \sum_{i=1}^{N} \theta_i x^{i-1}$$

Another approach that is often a better approximation for small N is:

$$\tilde{g}_{\theta_1,\dots,\theta_{2N+2}}(y, x) := \theta_{N+1} \log(x) + \theta_{N+2} \exp(x) + \sum_{i=1}^{N} \left(\theta_i x^{i-1} + \theta_{N+i} x^{-i}\right)$$

These approximation functions are not used as feature functions themselves. Since they are linear in $\boldsymbol{\theta}$ we can introduce a set of feature functions by using all summands separately as function in x and multiplying it by the indicator on the label:

$$\begin{aligned} f^{l}_{y'}(y, x) &= \mathbb{1}\{y = y'\} \log(x) \\ f^{e}_{y'}(y, x) &= \mathbb{1}\{y = y'\} \exp(x) \\ f^{\text{inv}}_{y',i}, (y, x) &= \mathbb{1}\{y = y'\} x^{-i} \\ f^{\text{mon}}_{y',i}, (y, x) &= \mathbb{1}\{y = y'\} x^{i} \end{aligned}$$

A complete set of approximating feature functions for a scalar observation x is given by a joined set of the above function templates.

Label relations

So far we described possibilities that allow for a systematic approximation and a model that takes care of every possible compatibility design between label and observations. The above functions are all used to define 'local' potentials, since they are not dealing with more than one label. The same way we introduced naive indicator for local potentials we can use them for network potentials as well. A corresponding set of functions is obtained by:

$$f_{y'_1,y'_2}(y_v, y_w) = \mathbb{1}\{y'_1 = y_v\}\mathbb{1}\{y'_2 = y_w\}$$

with a complete set of functions:

$$\mathcal{F}_3 := \{f_{y'_1,y'_2} | f_{y'_1 \in \mathcal{Y}, y'_2 \in \mathcal{Y}}\} \tag{2.8}$$

$$|\mathcal{F}_3| = |\mathcal{Y}|^2 \tag{2.9}$$

In the case of road quality as a label it is plausible to assume an ordinal structure of the label variable Y. In such cases it is also possible to define a smaller function set which uses a given metric on the difference of label values as compatibility measure, which can be handled in a single feature:

$$f(y_v, y_w) := d(y_v, y_w) \quad \text{with} \tag{2.10}$$

$$\text{'harmonic'} \quad d(x, y) := \exp(-|x - y|) \quad \text{or} \tag{2.11}$$

$$\text{'linear'} \quad d(x, y) := -|x - y| \quad \text{or} \tag{2.12}$$

$$\text{'directed'} \quad d(x, y) := x - y \quad \text{or} \tag{2.13}$$

$$\text{'dirichlet'} \quad d(x, y) := \mathbb{1}\{x = y\} \tag{2.14}$$

Further it is often of use to allow different label dependencies for different key observations. In the road quality model we want to model the quality label dependency for each functional class of road separately. This could be solved by using multiple templates for the potentials. In our single template situation we use an indicator on the key observations to distinguish between different classes:

$$f_{x'_1,x'_2,y'_1,y'_2}(x_v, x_w, y_v, y_w) = \mathbb{1}\{y'_1 = y_v\}\mathbb{1}\{y'_2 = y_w\}\mathbb{1}\{x'_1 = x_v\}\mathbb{1}\{x'_2 = x_w\}$$

The most general definition for a function class for label relation features is given by:

$$f^{(1)}_{\mathbf{x}'_1,\mathbf{x}'_2,y'_1,y'_2}(\mathbf{x}_v, \mathbf{x}_w, y_v, y_w) := \mathbb{1}\{\mathbf{x}'_1 = \mathbf{x}_v\}\mathbb{1}\{\mathbf{x}'_2 = \mathbf{x}_w\}\mathbb{1}\{y'_1 = y_v\}\mathbb{1}\{y'_2 = y_w\} \tag{2.15}$$

$$f^{(2)}_{y'_1,y'_2}(\mathbf{x}_v, \mathbf{x}_w, y_v, y_w) := d_x(x_v, x_w)\mathbb{1}\{y'_1 = y_v\}\mathbb{1}\{y'_2 = y_w\} \tag{2.16}$$

$$f^{(3)}_{\mathbf{x}'_1,\mathbf{x}'_2}(\mathbf{x}_v, \mathbf{x}_w, y_v, y_w) := \mathbb{1}\{\mathbf{x}'_1 = \mathbf{x}_v\}\mathbb{1}\{\mathbf{x}'_2 = \mathbf{x}_w\}d_y(y_v, y_w) \tag{2.17}$$

$$f^{(4)}(\mathbf{x}_v, \mathbf{x}_w, y_v, y_w) := d_x(\mathbf{x}_v, \mathbf{x}_w)d_y(y_v, y_w) \tag{2.18}$$

with d_x and d_y being a distance measure like the examples in 2.10.

2.5 Application

2.5.1 Feature function set

In practical application there are several strategies for determining a good set of features or feature functions. Using the above function definitions the problem is closely related to model selection in the context of linear modeling. There is the possibility of inducing feature functions iteratively, starting with a minimal set. This is called feature function induction and is described and evaluated for linear CRFs in [McC03]. Dealing with huge models we found the selection of good candidates for feature functions to add to the set of evaluated features, can be a computationally costly task. Therefore we make use of numerical sparsening techniques like L1 or L2 sparsening, as described in 4.4. In order for these methods to determine a suitable set of function we start out with a full set of feature functions and numerically determine features that are most irrelevant for the model. This is a strictly numerical procedure and highly dependent on the type of objective function (see chapter 4), but it is computationally inexpensive.

2.5.2 Feature normalization

Some of the numerical procedures we are going to introduce assume the feature functions to have the same range. Especially convergence criteria (5.5) and numerical model sparsening via penalty terms (4.4) require easily comparable model parameter dimensions. The straight forward approach to normalize the feature range is to transform all features in the form:

$$f_{k,\min} = \min_{(\mathbf{y}',\mathbf{x}')\in S} f_k(\mathbf{y}',\mathbf{x}') \quad \text{and} \quad f_{k,\max} = \max_{(\mathbf{y}',\mathbf{x}')\in S} f_k(\mathbf{y}',\mathbf{x}')$$

$$\tilde{f}_k(\cdot) = \frac{f_k(\cdot) - f_{k,\min}}{f_{k,\max}}$$

where S is a suitable set of label-observation pairs. In the function sets defined in the previous section this normalization is only necessary for the edge-distance features and the approximation of functional dependencies with continuous observations. In the case of edge-distance features it is expedient to normalize over all combinations of possible label realizations $\mathbf{y} \in \mathcal{Y} \times \mathcal{Y}$. In the functional approximation cases one has to carefully select the set of pairs S the function is normalized over. Such feature functions each depend on a certain dimension of the observation X_k. In many cases the empirical bounds within the dataset $S = (\mathbf{x}^{(0)}, \mathbf{y}^{(0)})$ are not a good choice for the normalization. Using the empirical bound causes two effects that result in unwanted normalization behaviour.

- If the dataset contains outliers, using the empirical bounds easily leads to a clustering of data which might render the feature invaluable.
- If the dataset is rather small and homogeneous it may not cover a representative range of the specific observation realizations

These issues can be drastically reduced by preprocessing the observed data. We propose to use a combination of data normalization followed up by feature function capping. Since the above problems only relate to the unbound functional approximations that deal with continuous observations we can make use of standard procedures used in statistics. Our proposed normalization procedure for a given continuous observation X_k with realizations $\mathbf{x}_k^{(0)}$ is:

- Normalization via z-Score
- Scaling with factor 1/2.68, for 2.68 is the boxplot-outlier definition for normal distributed variables
- Discard values > 1 as outliers
- Use monome base for function approximation. This directly implies $|f_i(\cdot)| \leq 1$

That way we receive a full set of features with $|f_i(\cdot)| \leq 1$, which ensures numerical stability and makes the model parameter $\boldsymbol{\theta}$ much better interpretable especially when using model sparsening that involves $\|\boldsymbol{\theta}\|$ like LASSO or ridge regression. From here on and especially we therefore assume $|f_i(\cdot)| \leq 1 \quad \forall i$.

2.6 Notation

In our applications we restrict the model definition to the one templated version of the general CRF, introduced in 2.3.1. However the distinction between local and edge-based feature functions is not necessary, due to the fact that a redefinition of the features allows to express an edge based feature as vertex feature (assigned to the adjacent vertices). For means of simplicity we therefore define:

$$\mathbf{y}_v := \{y_v | v \in N(v)\} \quad \text{and} \quad \mathbf{x}_v := \{x_v | v \in N(v)\}$$

This allows us to unify the local and edge-based feature functions as functions of $\mathbf{y}_v, \mathbf{x}_v$ and define the conditional distribution as factorization over V:

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z} \prod_{v \in V} \exp(\boldsymbol{\theta}' \mathbf{f}(\mathbf{y}_v, \mathbf{x}_v)) \quad \text{with} \quad Z = \sum_{\mathbf{y}'} \prod_{v \in V} \exp(\boldsymbol{\theta}' \mathbf{f}(\mathbf{y'}_v, \mathbf{x}_v)) \tag{2.19}$$

Now we gain a shorter notation of the log-likelihood function (single template version $N = 1$):

$$l(\boldsymbol{\theta}; \mathbf{y}^{(0)}, \mathbf{x}^{(0)}) = \ln \left(\frac{1}{Z} \prod_{v \in V} \exp(\boldsymbol{\theta}' \mathbf{f}(\mathbf{y}_v^{(0)}, \mathbf{x}_v^{(0)})) \right) = \sum_{v \in V} \boldsymbol{\theta}' \mathbf{f}(\mathbf{y}_v^{(0)}, \mathbf{x}_v^{(0)}) - \ln Z \tag{2.20}$$

And the gradient becomes

$$\frac{\partial}{\partial \boldsymbol{\theta}_k} l(\boldsymbol{\theta}; \mathbf{y}^{(0)}, \mathbf{x}^{(0)}) = \sum_{v \in V} f_k(\mathbf{y}_v^{(0)}, \mathbf{x}_v^{(0)}) - \frac{1}{Z} \sum_{\mathbf{y}'} \exp\left(\sum_{v \in V} \boldsymbol{\theta}' \mathbf{f}(\mathbf{y'}_v, \mathbf{x}^{(0)}) \right) \sum_{v \in V} f_k(\mathbf{y'}_v, \mathbf{x}^{(0)}) \tag{2.21}$$

$$= \sum_{v \in V} f_k(\mathbf{y}_v^{(0)}, \mathbf{x}_v^{(0)}) - \sum_{\mathbf{y}'} p(\mathbf{y}') \sum_{v \in V} f_k(\mathbf{y'}_v, \mathbf{x}^{(0)}) \tag{2.22}$$

$$= \mathbb{E}_0 \left[\sum_{v \in V} f_k(\mathbf{Y}_v, \mathbf{X}_v) \right] - \mathbb{E}_\infty \left[\sum_{v \in V} f_k(\mathbf{Y}_v, \mathbf{X}_v) \right] \tag{2.23}$$

$$\tag{2.24}$$

3 Inference

Statistical inference in the context of graphical models describes the process of calculating marginal probabilities for a trained model. Inference can be performed by deterministic approaches or Monte Carlo methods. In section 3.2 we introduce Belief Propagation, a commonly used deterministic approach, while the Markov Chain Monte Carlo (MCMC) approach is discussed in 3.3. Afterwards we introduce the so called Rao-Blackwellized Estimator for MCMC with Gibbs Sampling.

3.1 Inference in graphical models

In our work we will be using inference for two different purposes.

1. Inference is used to determine the most likely labelling $\mathbf{y_c}$ for a set of vertices of interest $\mathbf{c}$. This is done in order to use a trained model to draw conclusions regarding the labels from an observation data set $\mathbf{x}_c$.
2. We use inference to calculate vertex and edge marginals for maximum likelihood estimation. In chapter 4 we show that inference is needed for the training of the model via maximum likelihood.

In both scenarios we can make use of the standard approaches for CRFs.

3.2 Belief propagation

There are graph classes on which inference can be performed accurately and efficiently. For inference on tree structured graphs a class of algorithms called 'belief propagation' can be used. The procedure is explained in detail by Sutton and McCallum [SM10] in the context of inference for graphical models. The original paper by Pearl [Pea82] introduces this inference method for hierarchical structures or tree graphs. The basic idea works for all factor graphs and can be summarized by:

- The marginal distribution for the vertex of interest is given by product of contributions, called messages, from the adjacent factor nodes:

$$p_{X_v}(x_v) \propto \prod_{a \in N(v)} \mu_{a \to v}(x_v).$$

In order to be interpreted as probability the contributions have to be normalized

- The calculation of the messages for a factor node a is:

$$\mu_{a\to v}(x_v) = \sum_{\mathbf{x}'_a : x'_v = x_v} f_a(\mathbf{x}'_a) \prod_{v^* \in N(a)\setminus\{v\}} \mu_{v^*\to a}(x'_{v^*})$$

  $\mu_{v^*\to a}$ denotes the contribution of vertex v to the factor node a

- A message from a vertex node v to a factor node a is calculated by

$$\mu_{v\to a}(x_v) = \prod_{a^* \in N(v)\setminus\{a\}} \mu_{a^*\to v}(x_v)$$

For tree structured graphs the above message passing converges. For graphs containg loops there is the loopy belief propagation variant of this algorithm, which is not ensured to converge and is also not advised to be used for big graphs [SM10]. There are also approaches that combine the Loopy Belief Propagation and stochastic approaches (Markov Chain Monte Carlo), as described in [HRF06].

3.3 Markov Chain Monte Carlo

In many cases it is not computationally efficient to use deterministic approaches for the inference in graphical models. Stochastic approximation methods are a very popular alternative to Belief Propagation. Amongst the approximation methods the most used and most common approach is Markov Chain Monte Carlo. The idea of Monte Carlo methods is to implement the idea of approximating an expected value of a function of random variable $\mathbf{E}[f(\mathbf{X})]$ by the mean over a series of samples $\{\mathbf{x}_i\}_i$. In standard Monte Carlo approaches one uses the distribution of $\mathbf{X}$ to generate samples $\mathbf{x}_i$.
Markov Chain Monte Carlo methods however do not sample from the exact same unconditioned distribution for each sample. Rather one iteratively constructs a Markov Chain of random variables that the realizations are drawn from. Since we look at MCMC as an iterative process we can define the markov Chain to be indexed over $\mathbb{N}$. To ensure convergence it is necessary for the Markov Chain to have a stationary distribution $\pi(\cdot)$ which is fulfilled by all homogenic, aperiodic and irreducible Markov Chains.
This implies that after a suffiently long burn-in period the random samples will be drawn from distributions that approximate the stationary distribution $\pi(\cdot)$. For faster convergence we usually discard the first m samples since they are regarded as beeing bad approximations. This burn-in period is strongly dependent on the starting configuration and on $\pi(\cdot)$.

The target quantity can now be approximated in a similar way as Monte Carlo methods do. The theoretical justification for the approximation is given by the ergodic theorem (as

in [AG11]) instead of the law of large numbers as in Monte Carlo methods with independent sampling.

$$\mathbf{E}\left[f(X)\right] \approx \frac{1}{M} \sum_{j=m+1}^{M} f(X_j)$$

There are different ways to create sequences of samples which suffice the above conditions for the Markov Chain. A surprisingly simple and very popular sampler is the Metropolis-Hastings sampler ([Has70]). This approach samples according to a freely chosen distribution q and accepts or rejects samples by a likelihood quotient criterion. The algorithm in pseudo code can be seen in 3.1

Algorithm 3.1 Metropolis Hastings

1: Start with an (random) initial labelling $\mathbf{y}^{(0)}$
2: **while** not converged **do**
3: Sample $\mathbf{y}'$ using $q(\cdot|\mathbf{y}^{(i)})$
4: Sample a Uniform(0,1) random variable U
5: Calculate likelihood quotient $\alpha(\mathbf{y}', \mathbf{y}^{(i)}) = min\left(1, \frac{\pi(\mathbf{y}')q(\mathbf{y}^{(i)}|\mathbf{y}')}{\pi(\mathbf{y}^{(i)})q(\mathbf{y}'|\mathbf{y}^{(i)})}\right)$
6: **if** $U \leq \alpha(\mathbf{y}', \mathbf{y}^{(i)})$ **then**
7: $\mathbf{y}^{(i+1)} = \mathbf{y}'$
8: **else**
9: $\mathbf{y}^{(i+1)} = \mathbf{y}^{(i)}$
10: **end if**
11: **end while**

3.4 Gibbs Sampler

Gibbs sampling was introduced in 1984 by Geman and Geman [GG84] and is a special case of the Metropolis-Hastings sampler. It is often used for the sampling of random fields. Therefore the random variable of interest is indexed over the index set V as the set of vertices in the model definition of CRFs (2.19). The sampling algorithm makes use of the full conditional distribution of the stationary distribution $\pi(\cdot)$. The conditional distribution $q_v = \pi(y_v|\mathbf{y}_{-v})$ serves as sampling distribution at index location v. This yields a trivial acceptance coefficient:

$$\begin{aligned}\alpha(y_v, \mathbf{y}_{-v}, \mathbf{y}') &= min\left(1, \frac{\pi(\mathbf{y}')q(\mathbf{y}^{(i)}|\mathbf{y}')}{\pi(\mathbf{y}^{(i)})q(\mathbf{y}'|\mathbf{y}^{(i)})}\right) \\ &= \min\left(1, \frac{\pi(\mathbf{y}'_v|\mathbf{y}_{-i})q_v(\mathbf{y}_v|\mathbf{y}'_v, \mathbf{y}_{-v})}{\pi(\mathbf{y}_v|\mathbf{y}_{-i})q_v(\mathbf{y}'_v|\mathbf{y}_v, \mathbf{y}_{-v})}\right) = 1\end{aligned}$$

Since the distribution for CRFs is defined via the full conditionals 2.19 these probabilities are given by default and it is convenient to use Gibbs sampling. In our work we use Gibbs

Sampling to determine an a-posteriori marginal distribution for vertices and edges. Since the label values are restricted to a finite set $y_v \in \mathcal{Y}, \forall v \in V$, we can parametrize the a-posteriori distribution by the set of parameters $p_i, i \in \{1, \ldots, |\mathcal{Y}|\}$

$$p(y_v|\mathbf{y}_{\setminus v}, \mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{c \in \mathbf{C}_v} \exp(\boldsymbol{\theta}' \mathbf{f}(\mathbf{y}_c, \mathbf{x}_c)) \quad \text{with} \quad Z := \sum_{\mathbf{y}' \in \mathcal{Y}^{V_{\mathbf{C}_v}}} \exp(\theta \mathbf{f}(\mathbf{y}_c, \mathbf{x}_c))$$

After a series of MCMC iterations we receive a set of samples of the random field which is used to estimate the marginal distribution's parameters.

Algorithm 3.2 MCMC algorithm with Gibbs sampling

1: Start with an (random) initial labelling $\mathbf{y}^{(0)}$
2: **while** not converged **do**
3: **for each** $v \in V$ **do**
4: Sample $y_v^{(i+1)}$ using $p(y_v^{(i+1)}|\mathbf{y}_{<v}^{(i+1)}, \mathbf{y}_{>v}^{(i)}, \mathbf{x})$ ▷ Gibbs Sampling
5: **for each** $\tilde{y} \in \mathcal{Y}$ **do**
6: $\hat{P}(Y_v = \tilde{y}) = \hat{p}_{v,\tilde{y}} := \frac{1}{|\mathcal{S}|} \sum_{\mathbf{y} \in \mathcal{S}} \mathbb{1}(y_v = \tilde{y})$ ▷ Calculate estimator
7: **end for**
8: **end for**
9: check for convergence
10: increase iteration counter $i = i + 1$
11: **end while**

3.5 Estimators

3.5.1 Monte Carlo estimator

After i iterations, the MCMC has generated a set of samples $\mathcal{S} := \{\mathbf{y}^{(i)}\}_i$. Then the Monte Carlo estimator of the a-posteriori distribution parameters for the marginal probability at vertex v is defined by:

$$P(Y_v = \tilde{y}) = E\left[\mathbb{1}(Y_v = \tilde{y})\right] \approx \hat{p}_{v,\tilde{y}} := \frac{1}{|\mathcal{S}|} \sum_{\mathbf{y} \in \mathcal{S}} \mathbb{1}(y_v = \tilde{y})$$

The sample set $\mathcal{S}$ is iteratively constructed by using Gibbs sampling (3.4). The procedure is shown in algorithm 3.2. In the pseudo-code we demonstrated the algorithm for vertex-based marginals. The edge-based marginal distributions can be estimated equivalently. A general formulation for an estimator for marginal probabilities on a set of nodes $c \subseteq V$ of the CRF using a set of samples is given by:

$$P(\mathbf{Y}_c = \mathbf{y'}_c) = E\left[\mathbb{1}(\mathbf{Y}_c = \mathbf{y'}_c)\right] \approx \hat{p}_{c,\mathbf{y'}_c} := \frac{1}{|\mathcal{S}|} \sum_{\mathbf{y} \in \mathcal{S}} \mathbb{1}(y_c = \mathbf{y'}_c)$$

3.5.2 Rao-Blackwellized estimator

Basic Idea: If we are looking for the scalar marginal distributions, we can use a more efficient estimator for the stationary distribution. The idea is to use the full conditional distributions that were used to create the sample set as estimators for the stationary distribution.

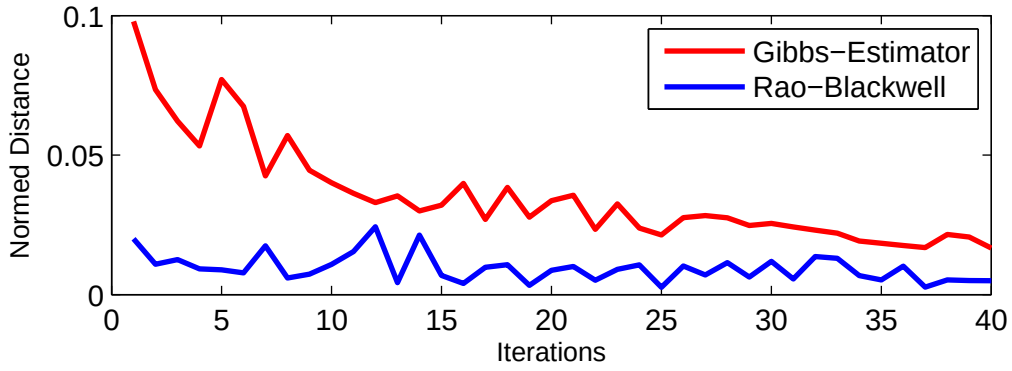
$$P(Y_v = \tilde{y}) = E\left[\mathbb{1}(Y_v = \tilde{y})\right] \approx \hat{r}_{v,\tilde{y}} := \frac{1}{|\mathcal{S}|} \sum_{\mathbf{y} \in \mathcal{S}} E(Y_v = \tilde{y} | \mathbf{y}_{-v})$$

Since $y_v \sim Y_v | \mathbf{y}_{-v}$ the definition of $\hat{r}_{v,\tilde{y}}$ satisfies the conditions of Rao-Blackwell theorem and therefore

$$\mathrm{MSE}(\hat{r}_{v,\mathbf{y'}}) \leq \mathrm{MSE}(\hat{p}_{c,\mathbf{y'}})$$

Using the local full conditionals and its corresponding expected values usually results in much faster convergence rates. The scale of improvement over the standard estimator cannot be determined analytically, since it is highly dependent on the marginal distribution and therefore on the complete model setup. We conducted a series of simulations that show the improvement in a set of sample situations. One example is depicted in figure 3.1. In this particular simulation we used a lattice graph of 10 by 10 vertices and evaluated the normalized differences of a single vertex' marginal distribution of sequential iterations. The results show that the Rao-Blackwellized estimator performs much better and does not improve after a few iterations. However this is highly problem-dependent and differs slightly when using other convergence criteria.

Figure 3.1: Simulation results showing normed differences of distributions of a single vertex in the random field



The Rao-Blackwellized estimator $\hat{r}$ however is only directly applicable to the full conditional probabilities and only useful in the case one wants to estimate the scalar marginals at a single vertex $v \in V$. Especially in the context of model training we benefit from also efficiently estimating the marginal probabilities for the edges of a CRF.
Let now denote $\mathbf{e} = (v, w) \in E$ an edge. The marginal probabilities for this edge can be calculated using the set of samples from the MCMC sampling and the standard estimator $\hat{p}_{e,\mathbf{y'}_e}$. Following the idea of the Rao-Blackwellized estimator for the vertex marginals, we propose another estimator that is suitable for efficient estimation of the edge-based marginals:

$$\begin{aligned} P(Y_e = \mathbf{y'}_e) &= P((Y_u, Y_v) = (a, b)) = P(Y_u = a | Y_v = b) P(Y_v = b) \\ &\approx \frac{1}{m} \sum_{j=1}^{m} \hat{r}_{u,a} \cdot \mathbb{1}(y_v = b) =: \hat{s}_{u,v}(\mathbf{y}) \end{aligned}$$

Here $\hat{r}_{u,a}$ is the Rao-Blackwellized estimator from above. However $\hat{s}_{u,v}(\mathbf{y})$ is not really Rao-Blackwellized, since the joined conditional marginals of vertices remains. We can now increase the gain from the sample set $\mathbf{y}$ by using both conditional estimators: $\hat{s}_{u,v}(\mathbf{y})$ and $\hat{s}_{v,u}(\mathbf{y})$. For large sample sets the differnce between these estimators can be neglected, but for smaller sample sizes we can use a mixed estimator to obtain a faster convergence. Therefore we use the number of occurrences as weighting coefficient, since it correlates with the relative reliabilities of the estimators.

$$\begin{aligned} N_u(a) &:= \sum_{j=1}^{m} \mathbb{1}(y_u = a) \quad \text{and} \quad N_v(b) := \sum_{j=1}^{m} \mathbb{1}(y_v = b) \\ \alpha &:= \frac{N_u(a)}{N_u(a) + N_u(b)} \\ \hat{r}_{u,v,a,b} &:= \alpha \hat{s}_{u,v}(\mathbf{y}) + (1 - \alpha) \hat{s}_{v,u}(\mathbf{y}) \end{aligned}$$

Although the estimator for the edge marginals is not proven to be the most efficient we achieve a great gain in efficiency compared to the standard Monte Carlo estimator.

3.6 Convergence

In order to find a suitable convergence criterion one has to bear in mind that the convergence of the estimators in algorithm 3.2 are relying on stochastic sampling. This might lead to spurious convergence when using convergence measures from deterministic theory. A different initialization of the random number generator or a different initial labelling $\mathbf{y}^{(0)}$ might lead to vastly different estimations for the marginal distributions for a long period of iterations. However the variation of the a-posteriori parameter estimator $\hat{p}$ is decreasing almost monotonically in many cases. Especially if $|\mathcal{Y}|$ is small and if the conditional probabilities that are used by the Gibbs sampler are of low variance, this might lead to false convergence assumptions.
Following the ideas of Gelman and Rubin ([GR92]) we execute the same MCMC algorithm multiple times with different initial labelling $\mathbf{y}^{(0)}$ and monitor the variation of the different estimations of the parameters. This approach has been quoted as the "Within and In-Between Criterion" and can be found in [GRS95]. In order to define the needed quantities let $\hat{p}_{v,\tilde{y}}^{(i,j)}$ be the estimation of iteration j, of MCMC execution number i, at vertex v and for the value $\tilde{y}$. Given a set of m parallel MCMC executions and n iterations we go ahead and define the In-Between-Variance B and Within-Sequence-Variance W as follows:

$$B_{v,\tilde{y}} = \frac{n}{m-1}\sum_{i=1}(\bar{p}_{v,\tilde{y}}^{(i)} - \bar{p}_{v,\tilde{y}})^2 \quad \text{with} \quad \bar{p}_{v,\tilde{y}}^{(i)} = \frac{1}{n}\sum_{j=1}^{n}\hat{p}_{v,\tilde{y}}^{(i,j)} \quad \text{and} \quad \bar{p}_{v,\tilde{y}} = \frac{1}{m}\sum_{i=1}^{m}\bar{p}_{v,\tilde{y}}^{(i)}$$

$$W_{v,\tilde{y}} = \frac{1}{m}\sum_{i=1}^{m}\left(s_{v,\tilde{y}}^{(i)}\right)^2 \quad \text{with} \quad s_{v,\tilde{y}}^{(i)} = \frac{1}{n-1}\sum_{j=1}^{n}(p_{v,\tilde{y}}^{(i,j)} - \bar{p}_{v,\tilde{y}}^{(i)})^2$$

One can now use a combined variance estimate that is unbiased under stationarity assumption:

$$\hat{\text{var}}(\hat{p}_{v,\tilde{y}}) := \frac{n-1}{n}W_{v,\tilde{y}} + \frac{1}{n}B_{v,\tilde{y}}$$

and this leads to the following convergence measure:

$$\sqrt{\hat{R}_{v,\tilde{y}}} = \sqrt{\frac{\hat{\text{var}}(\hat{p}_{v,\tilde{y}})}{W_{v,\tilde{y}}}}$$

Convergence is reached when the $\sqrt{\hat{R}_{v,\tilde{y}}}$ is approximately 1. Gelman and Rubin also provide suggestions on how to measure converge of a set of predictors. Since in our case the set of predictors is rather big (magnitude of $|\mathcal{Y}| \cdot |V|$), we define the convergence measure as:

$$R := \max_{v\in V,\tilde{y}\in\mathcal{Y}}\sqrt{\hat{R}_{v,\tilde{y}}}$$

3.7 Small adjustments to the algorithm

The naive Gibbs Sampler algorithm from 3.4 often leads to slow converge due to the fact, that the samples from early iterations are often further away from samples of the a-posteriori than later samples. A first brute force way to deal with that problem is to discard a fixed number of samples from the first iterations. These early iteration are therefore called the "burn-in period". Other approaches additionally introduce behaviour like simulated annealing. We cope with the problem of bad initial samples together with the problem of slowly converging estimators. To achieve this, we define our Monte Carlo estimator to integrate only over the last N samples:

$$\hat{p}_{v,\tilde{y}} := \frac{1}{N} \sum_{i=1}^{N} \mathbb{1}(y_v^{(i)} = \tilde{y})$$

The backdraw of this approach is a limited accuracy of the estimators. This makes choosing a suitable window size N essential. Slowly converging series of estimations also limits the power of our convergence criterion $R := \max_{v \in V, \tilde{y} \in \mathcal{Y}} \sqrt{\hat{R}_{v,\tilde{y}}}$. The estimator for the In-Between-Variance B is only very slowly reacting to a converging series. Therefore we also introduce a window size L in the estimation of the variances:

$$B^L_{v,\tilde{y}} = \frac{L}{m-1} \sum_{i=1}^{m} (\bar{p}^{(i)}_{v,\tilde{y}} - \bar{p}_{v,\tilde{y}})^2 \quad \text{with} \quad \bar{p}^{(i)}_{v,\tilde{y}} = \frac{1}{L} \sum_{j=1}^{L} \hat{p}^{(i,(n+1)-j)}_{v,\tilde{y}} \quad \text{and} \quad \bar{p}_{v,\tilde{y}} = \frac{1}{m} \sum_{i=1}^{m} \bar{p}^{(i)}_{v,\tilde{y}}$$

$$W^L_{v,\tilde{y}} = \frac{1}{m} \sum_{i=1}^{m} \left(s^{(i)}_{v,\tilde{y}} \right)^2 \quad \text{with} \quad s^{(i)}_{v,\tilde{y}} = \frac{1}{n-1} \sum_{j=1}^{L} (p^{(i,(n+1)-j)}_{v,\tilde{y}} - \bar{p}^{(i)}_{v,\tilde{y}})^2$$

3.8 Simulation study

In a small simulation study we show how the different parameters of the convergence monitoring work out in practical application. Especially the choice of the parameter L in the definition for $B^L_{v,\tilde{y}}$ and $W^L_{v,\tilde{y}}$ shall be researched. The algorithm that is used for sampling is depicted in 3.3.

The underlying graphical model is a small lattice graph of dimensions 10 by 10. The complete setup, together with the parameter choices is shown in table 3.8.

Conclusion:
The selected results shown in 3.2 confirm the intuition that the amount of samples for the estimation N has to exceed the amount of samples for the variance estimation L in order for this measure to reach convergence. This is however highly dependent on the graph itself. Using non lattice graphs and higher amount of vertices changes the convergence behaviour. Essentially using a convergence measure for gibbs sampling convergence is an open problem and we are not aware of a definition that works equally for all possible problem setups. In the CRF application cases for the road roughness extrapolation we

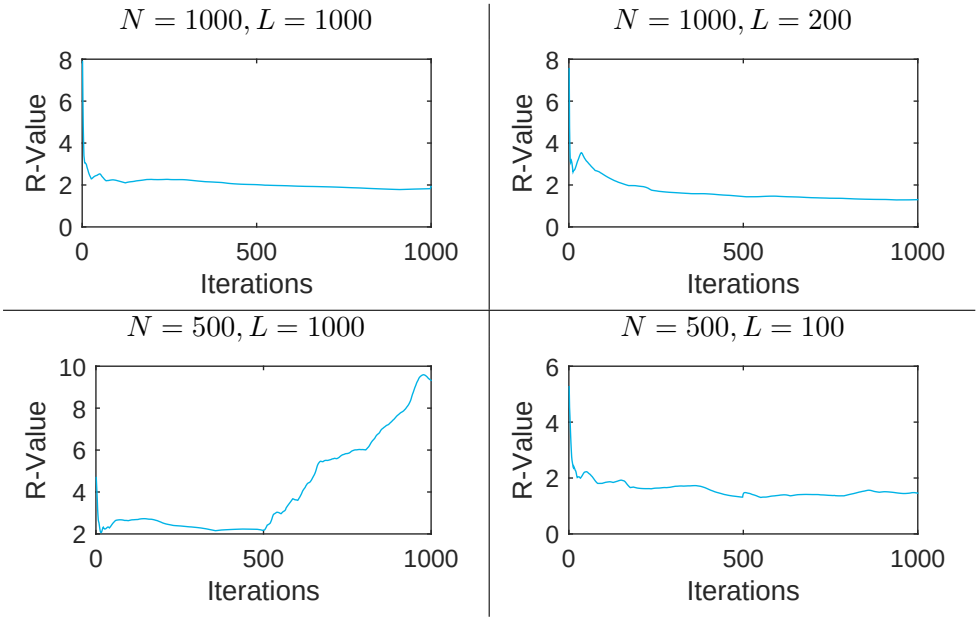
Algorithm 3.3 MCMC variant that was used for the study

Start with (random) initial labeling $\mathbf{y}^{(i,0)}$ with $1 \leq i \leq m$
while not converged **do**
  for $i = 1, \ldots, m$ **do** ▷ Iteration over parallel executions
    for each $v \in V$ **do**
      Sample $y_v^{(i,(j+1))}$ using $p(y_v^{(i,j+1)} | \mathbf{y}_{<v}^{(i,j+1)}, \mathbf{y}_{>v}^{(i,j)}, \mathbf{x})$ ▷ Gibbs Sampling
      for each $\tilde{y} \in \mathcal{Y}$ **do**
        Limit Window size: $S = min(j, N)$
        $\hat{p}_{v,\tilde{y}}^{(i,j+1)} := \frac{1}{S} \sum_{k=1}^{S} \mathbb{1}(y_v^{(k)} = \tilde{y})$ ▷ Calculate estimator
      end for
    end for
  end for
  Calculate limited window size variances:$W_{v,\tilde{y}}^{L}$ and $B_{v,\tilde{y}}^{L}$
  Taking these variances, calculate $R = \max_{v \in V, \tilde{y} \in \mathcal{Y}} \sqrt{\hat{R}_{v,\tilde{y}}}$
  if $R < \epsilon$ **then**
    converged = true
  end if
  increase iteration counter $j = j + 1$
end while

Table 3.1: Simulation study - settings

Number of Vertices	n	10 x 10
Number of Iterations	max(j)	10000
Target dimension	$\lvert\mathcal{Y}\rvert$	5
Feature functions		3
Parallel Executions	m	4
Estimation Window	N	{50,100,200,500,1000}
Variance Window	L	{50,100,200,500,1000}

Table 3.2: Simulation study - selected results



advise to choose a certain acceptable number of iterations or computation time and stop the iteration. Especially when using the Rao-Blackwellized estimator (as in 3.5.2)this delivers good results. In chapter 7 we use MCMC for inference on CRFs with a fixed number of 20 iterations.

4 Training

Training graphical models can be performed as supervised or unsupervised training task, depending on the available data and classification target. Since we focus on extrapolation tasks of standardized quantities like the IRI or the ISO roughness index, we use supervised learning techniques. Therefore the task is to find the best set of model parameters $\boldsymbol{\theta}^*$ to fit the model to a given training dataset $S = (\mathbf{y}^{(0)}, \mathbf{x}^{(0)})$, where both, observations and the classification of the data is available. For conditional random fields, training is usually performed by using maximum likelihood methods. In section 4.1 we demonstrate the issues that occur when training a CRF and methods to cope with them. We proceed by demonstrating an alternative training method called contrastive divergence in 4.2. We also show how pseudo likelihood objective functions can enhance the training and how to sparse the model. We concentrate on the special case of the previously introduced single template CRF with edge-based and vertex-based potentials.

4.1 Maximum Likelihood

Maximizing the random fields likelihood function on a training dataset is the most common way to train a graphical model. In this section we will introduce the classical maximum likelihood estimation, discuss it and introduce variants and alternatives for big graphical models. Without loss of generality we can assume to use a training dataset that consists of a single graph. This is possible due to the fact that we are training a single template version(2.3.2) for the whole graph. Since we never require this graph to consist of only one component we can just treat a training set with multiple samples of one graph as a single sample with multiple components. We will start by reviewing the log-likelihood function and its gradient for a general CRF model. This has been introduced already in [2.21,2.20]. Given a training dataset $S = (\mathbf{y}^{(0)}, \mathbf{x}^{(0)})$ the log likelihood for the CRF is:

$$l(\boldsymbol{\theta}; \mathbf{y}^{(0)}, \mathbf{x}^{(0)}) = \sum_{v \in V} \boldsymbol{\theta}' \mathbf{f}(\mathbf{y}_v^{(0)}, \mathbf{x}_v^{(0)}) - \ln Z \quad \text{with} \quad Z := \sum_{\mathbf{y}'} \exp\left(\sum_{v \in V} \boldsymbol{\theta}' \mathbf{f}(\mathbf{y}'_v, \mathbf{x}_v^{(0)}) \right) \tag{4.1}$$

and its gradient:

$$\frac{\partial}{\partial \boldsymbol{\theta}_k} l(\boldsymbol{\theta}; \mathbf{y}^{(0)}, \mathbf{x}^{(0)}) = \sum_{v \in V} f_k(\mathbf{y}_v^{(0)}, \mathbf{x}_v^{(0)}) - \sum_{\mathbf{y}'} p(\mathbf{y}') \sum_{v \in V} f_k(\mathbf{y}'_v, \mathbf{x}^{(0)}) \tag{4.2}$$

In the likelihood function 4.1 as well as the gradient 4.2 we find a sum over realizations of $\mathbf{Y}$. This is intractable in general due to the high dimensionality of $\mathbf{y}$ in graphical models. Although the derivative can be simplified to a far smaller summation term, the intractability problem remains.

$$\frac{\partial}{\partial \boldsymbol{\theta}_k} l(\boldsymbol{\theta}; \mathbf{y}^{(0)}, \mathbf{x}^{(0)}) = \sum_{v \in V} f_k(\mathbf{y}_v^{(0)}, \mathbf{x}_v^{(0)}) - \sum_{v \in V} \sum_{\mathbf{y'}_v \in \mathcal{Y}^{\mathrm{N}(v)}} p(\mathbf{y'}_v) f_k(\mathbf{y'}_v, \mathbf{x}^{(0)}) \tag{4.3}$$

The difficulty of this simplified version of the gradient relies on the difficulty of calculating $p(\mathbf{y'}_v)$ for every set of neighbors $\mathrm{N}(v)$ of all vertices $v \in V$. This means that for the evaluation of the gradient one has to calculate the marginal distributions for the set of neighborhoods in the graph. This is a classical inference task and in chapter 3 we discuss ways to approximate those marginals. For CRFs on big graphs using MCMC methods with Gibbs sampling is known to be the inference method of choice. Iteratively approaching the maximum of the likelihood function 4.1 however is of great computational complexity since the MCMC used for approximating the gradient converges rather slowly in general and leads to objective functions that vary each iteration with high variation. Our Rao-Blackwellized MCMC estimators improve on that issue but also meet their limits in practical application. The next sections will present different ways to cope with those intractabilities by defining approximate solutions to maximizing the likelihood.

4.2 Contrastive divergence

The method of contrastive divergence focuses on an equivalent problem to maximizing the likelihood function. Before going into detail we define the Kullback–Leibler divergence:

Definition 4.2.1 (Kullback–Leibler divergence)**.** *Let $\mathbb{P}_1, \mathbb{P}_2$ be random measures with supp$(\mathbb{P}_2) \subseteq$ supp$(\mathbb{P}_1) = \mathcal{A}$. The Kullback–Leibler divergence is defined as:*

$$D_{KL}(\mathbb{P}_1 || \mathbb{P}_2) := \sum_{A \in \mathcal{A}} \ln\left(\frac{\mathbb{P}_1(A)}{\mathbb{P}_2(A)}\right) \mathbb{P}_1(A) = \sum_{A \in \mathcal{A}} \ln\left(\mathbb{P}_1(A)\right) \mathbb{P}_1(A) - \sum_{A \in \mathcal{A}} \ln(\mathbb{P}_2(A)) \mathbb{P}_1(A)$$

This is an asymmetric measure for the inequality of two distributions and is first described in [KL51].
The method of minimizing contrastive divergence can be applied to a wide variety of models, including generative and discriminative graphical models. Therefore we will just make the model assumption that the probability distribution of the model has the form of a Gibbs model.

$$p(\mathbf{x}; \boldsymbol{\theta}) = \frac{1}{Z(\boldsymbol{\theta})} \exp(-E(\mathbf{x}; \boldsymbol{\theta}))$$

where $\boldsymbol{\theta} \in \mathbb{R}^n$ is a parameter vector and $Z(\boldsymbol{\theta}) = \sum_x \exp(-E(\mathbf{x}; \boldsymbol{\theta}))$ is a normalization constant. The function $E(\cdot; \boldsymbol{\theta})$ is an energy function that is of an exponential nature

in the case of Markov fields (due to the Hammersley–Clifford theorem 2.1.4). The log-likelihood function of such a model (Markov random field), given a set of samples $\{\mathbf{x}^{(n)}\}_n$ is:

$$l(\boldsymbol{\theta}; \mathbf{x}) = \mathbf{E}\left[E(\mathbf{x}_i; \boldsymbol{\theta})\right]_0 + \ln Z(\boldsymbol{\theta})$$

The gradient can be written as

$$\frac{\partial l(\boldsymbol{\theta}; \mathbf{x})}{\partial \boldsymbol{\theta}} = -\mathbf{E}\left[\frac{\partial E(\mathbf{x}; \boldsymbol{\theta})}{\partial \boldsymbol{\theta}}\right]_0 + \mathbf{E}\left[\frac{\partial E(\mathbf{x}; \boldsymbol{\theta})}{\partial \boldsymbol{\theta}}\right]_\infty$$

Here $\mathbf{E}\left[\cdot\right]_0$ denotes the expected value under the observed distribution, while $\mathbf{E}\left[\cdot\right]_\infty$ denotes the expected value under the distribution $\mathbb{P}(\mathbf{X}; \boldsymbol{\theta})$. The latter term is intractable in general. Hinton [Hin00] proposed to look at the constrastive divergence as a measure for convergence.

Definition 4.2.2 (Constrastive Divergence)**.** *Let $\{\mathbb{P}_n\}_{n\in\mathbb{N}}$ be a sequence of random measures with equal support that converges in probability to $\mathbb{P}_\infty$. The constrastive divergence of order n is defined by:*

$$CD_n := D_{KL}(\mathbb{P}_0||\mathbb{P}_\infty) - D_{KL}(\mathbb{P}_n||\mathbb{P}_\infty)$$

Minimizing the contrastive divergence of order n requires to know its gradient:

$$-\frac{\partial}{\partial \boldsymbol{\theta}}(CD_n) = \mathbf{E}\left[\frac{\partial E(\mathbf{x}; \boldsymbol{\theta})}{\partial \boldsymbol{\theta}}\right]_0 - \mathbf{E}\left[\frac{\partial E(\mathbf{x}; \boldsymbol{\theta})}{\partial \boldsymbol{\theta}}\right]_n + \frac{\partial \mathbb{P}_1}{\partial \boldsymbol{\theta}} \frac{\partial D_{\mathrm{KL}}(\mathbb{P}_1||\mathbb{P}_\infty)}{\partial \mathbb{P}_1}$$

Hinton shows empirically [Hin00] that the third term on the right hand side can be usually be ignored for model training. This motivates the final formulation of the principle of contrastive divergence based learning. The direction of the parameter updates due to CD_n training is given by:

$$\Delta\boldsymbol{\theta} \propto \mathbf{E}\left[\frac{\partial E(\mathbf{x}; \boldsymbol{\theta})}{\partial \boldsymbol{\theta}}\right]_0 - \mathbf{E}\left[\frac{\partial E(\mathbf{x}; \boldsymbol{\theta})}{\partial \boldsymbol{\theta}}\right]_n$$

4.2.1 CD_n-Training of CRFs

Training through minimizing Contrastive Divergence as it was proposed in the previous paragraph is capable of training graphical models in general. In the case of conditional random fields with a single template (as introduced in 2.3.2) the training term becomes more distinct. The Gibbs energy function E in the CRF case is defined as:

$$E(\mathbf{y}, \mathbf{x}; \boldsymbol{\theta}) = \sum_v \boldsymbol{\theta}' \mathbf{f}(\mathbf{y}_v, \mathbf{x}_v)$$

where $\mathbf{f}$ denotes the vector of feature functions as described in 2.4.2. Since we focus on the single template version of CRFs we always train the model using a single sample. As mentioned in the introduction of this chapter we can treat training sets with multiple

training samples the exact same way. The standard distribution estimator, we are referring to as the observed distribution, basing on the observations $\mathbf{y}^{(0)}, \mathbf{x}^{(0)}$ is given by:

$$p(\mathbf{y}|\mathbf{x}^{(0)}) = \mathbb{1}(\mathbf{y} = \mathbf{y}^{(0)})$$

Analogously the result of n MCMC iterations $\{\mathbf{y}^{(i)}, \mathbf{x}^{(0)}\}_{i \leq n}$ is the observed distribution

$$p(\mathbf{y}|\mathbf{x}^{(0)}) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}(\mathbf{y} = \mathbf{y}^{(i)})$$

The expected values of the MCMC generated probabilities are

$$\mathbf{E}\left[\frac{\partial E(\mathbf{y}; \mathbf{x}^{(0)}, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}}\right]_n = \frac{1}{n} \sum_{i=1}^{n} \frac{\partial E(\mathbf{y}^{(i)}; \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \frac{1}{n} \sum_{i=1}^{n} \sum_{v} \mathbf{f}(\mathbf{y}_v^{(i)}, \mathbf{x}_v^{(0)})$$

Therefore the direction of the parameter updates in the CRF training by minimizing CD_n are given by:

$$\Delta\boldsymbol{\theta} \propto \frac{1}{n} \sum_{i=1}^{n} \sum_{v} (\mathbf{f}(\mathbf{y}_v^{(0)}, \mathbf{x}_v^{(0)}) - \mathbf{f}(\mathbf{y}_v^{(i)}, \mathbf{x}_v^{(0)}))$$

4.2.2 Enhanced CD_n-Training

In practice the MCMC estimator as used in 4.2.1 is asymptotically correct, but especially in the training with contrastive divergence the number of MCMC samples is usually chosen to be very low in order to reduce computation cost. In the context of CRFs we can assume $|\mathcal{Y}|^{|V|}$ to be very big and especially that the number of MCMC samples is relatively small $n \ll |\mathcal{Y}|^{|V|}$. This leaves most realizations $\mathbf{y}$ indistinguishable regarding their probability since it is very likely that no realization has occurred more than once during MCMC.

$$p(\mathbf{y}|\mathbf{x}^{(0)}) \approx \begin{cases} \frac{1}{n} & \text{if } \mathbf{y} \text{ occured during MCMC} \\ 0 & \text{else} \end{cases}$$

Using this empirical distribution results in a rather bad approximation of $\mathbf{E}[\cdot]_\infty$ for small numbers of MCMC iterations and big graphs. We can improve on that by using the MCMC samples together with their Gibbs energy. This can be motivated by looking at the original definition of conditional probability for CRFs. We can use the MCMC samples to calculate an approximation for the intractable partition function directly approximate the probability of the CRF:
Let Z_n be the MCMC approximations partition function with

$$Z(\mathbf{x}) = \sum_{\mathbf{y}'} \prod_{v \in V} \exp(\boldsymbol{\theta}' \mathbf{f}(\mathbf{y}'_v, \mathbf{x})) \approx \sum_{i=1}^{n} \prod_{v \in V} \exp(\boldsymbol{\theta}' \mathbf{f}(\mathbf{y}_v^{(i)}, \mathbf{x}^{(0)})) =: Z_n(\mathbf{x}^{(0)})$$

then the probability approximation is given by:

$$\begin{aligned}
p(\mathbf{y}|\mathbf{x}) &= \frac{1}{Z(\mathbf{x})} \prod_{v \in V} \exp(\boldsymbol{\theta}' \mathbf{f}(\mathbf{y}_v, \mathbf{x}_v)) \\
&\approx \left[\frac{1}{Z_n(\mathbf{x})} \prod_{v \in V} \exp\left(\boldsymbol{\theta}' \mathbf{f}(\mathbf{y}_v, \mathbf{x}_v)\right) \right] \sum_{i=1}^{n} \mathbb{1}(\mathbf{y} = \mathbf{y}^{(i)}, \mathbf{x} = \mathbf{x}^{(0)}) \\
&= \sum_{i=1}^{n} \frac{1}{Z_n(\mathbf{x})} \prod_{v \in V} \exp\left((\boldsymbol{\theta}' \mathbf{f}(\mathbf{y}_v^{(i)}, \mathbf{x}_v^{(0)}) \right) \mathbb{1}(\mathbf{y} = \mathbf{y}^{(i)}, \mathbf{x} = \mathbf{x}^{(0)}) \\
&= \sum_{i=1}^{n} \omega_i \mathbb{1}(\mathbf{y} = \mathbf{y}^{(i)}, \mathbf{x} = \mathbf{x}^{(0)}) \\
&=: p_n(\mathbf{y}|\mathbf{x})
\end{aligned}$$

Here $\omega_i := \frac{1}{Z_n(\mathbf{x}^{(0)})} \prod_{v \in V} \exp\left(\boldsymbol{\theta}' \mathbf{f}(\mathbf{y}_v^{(i)}, \mathbf{x}_v^{(0)}) \right)$ describes the weights for the local samples by their Gibbs energies. Further we note:

$$\omega_i \overset{n \to \infty}{\to} p(\mathbf{y}_i|\mathbf{x})$$

This is similar to the Rao–Blackwellized marginal estimator in 3.5.2, differing in the way of normalizing. The marginal probabilities can be normalized by the sum over local environments while ω_i uses graph-wide normalization. Besides an improved way of interpreting the MCMC samples this approach also yields a consistent interpretation as an approximate gradient descent method. The objective function is given by the log-likelihood function of the MCMC based empirical distribution defined above:

$$l(\boldsymbol{\theta}; \mathbf{y}^{(0)}, \mathbf{x}^{(0)}) = \ln p_n(\mathbf{y}^{(0)}|\mathbf{x}^{(0)}) = \sum_{v \in V} \boldsymbol{\theta}' \mathbf{f}(\mathbf{y}_v^{(0)}, \mathbf{x}_v^{(0)}) - \ln Z_n$$

The gradient for the log-likelihood function is given by:

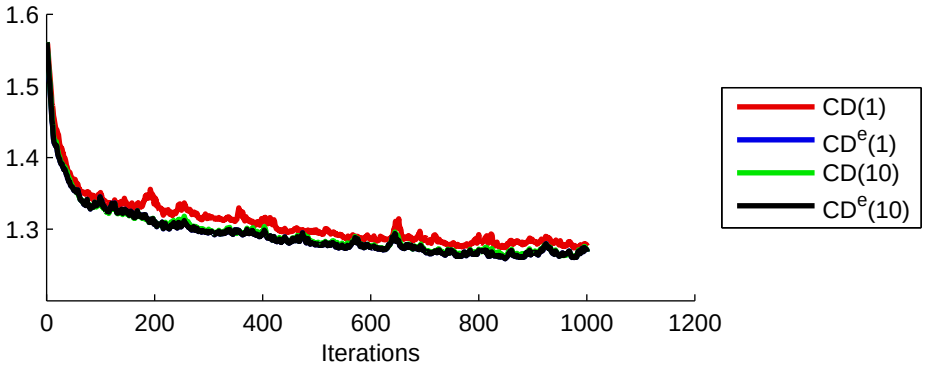
$$\begin{aligned}
\frac{\partial}{\partial \boldsymbol{\theta}} l(\boldsymbol{\theta}; \mathbf{y}^{(0)}, \mathbf{x}^{(0)}) &= \sum_{v \in V} \mathbf{f}(\mathbf{y}^{(0)}; \mathbf{x}^{(0)}) - \sum_{v \in V} \sum_{i=1}^{n} p_n(\mathbf{y}^{(i)}; \mathbf{x}^{(0)}) \mathbf{f}(\mathbf{y}^{(i)}; \mathbf{x}^{(0)}) \\
&= E_{p_0} \left[\sum_{v \in V} \mathbf{f}(\mathbf{Y}_v; \mathbf{X}_v) \right] - E_{p_n} \left[\sum_{v \in V} \mathbf{f}(\mathbf{Y}_v; \mathbf{X}_v) \right]
\end{aligned}$$

For a fixed set of MCMC iterations, this approach allows us to make use of standard stepsize choosing methods that are available for gradient based methods. As notation for the training of CRFs using our proposed variant of the contrastive divergence minimization method we will use CD_n^e as abbreviation for 'enhanced contrastive divergence minimization'.

4.2.3 Comparison of CD and enhanced CD

In order to measure the benefit we gain from using the enhanced version of the contrastive divergence method, we conducted a series of simulation experiments. The complete results can be found in the appendix (9.4). This also includes the optimization over random subsamples of the graph in order to speed up the optimization. The section 5.3 covers the main ideas. Figure 4.1 shows a prototypical example of the results. The quantity batchsize refers to the size of the subsample of the graph that has been used for the optimization. This study let to the following conclusions:

Figure 4.1: Contrastive divergence and enhanced method: stepsize 0.1, batchsize 10 , number of MCMC samples in brackets



- Enhanced CD is faster in convergence and is of less stochastic variation
- CD convergence behavior improves if more MCMC runs are used
- This effect of using more MCMC runs is very small when using enhanced algorithm, almost not present

This leads to the conclusion that the use of the enhanced CD is always encouraged. Further in the CRF context it is not efficient to increase on the number of MCMC iterations. Within the scope of CD the method of choice is 'CD_1^e'.

4.2.4 Linesearch with approximated objective functions

The method of minimizing contrastive divergence uses an approximation of the CD_n term as an approximation to the log-likelihood function. Since every iteration this approximation is redone for an updated set of parameters $\boldsymbol{\theta}^{(i)}$ the objective function is of random nature. This yields the problem that typical approaches to finding a valid stepsize $\boldsymbol{\alpha}$ are not applicable. This problem becomes smaller with higher values of n but cannot be circumvented. For $n \to \infty$ the approximated likelihood converges to the real likelihood function and a therefore a robust objective function. In such cases a linesearch algorithm

using a backtracking that ensures the Wolfe-condition 5.2.1 can be used. However in the context of CRFs we usually choose $n \ll \infty$ which renders classic stepsize adaptation invalid. Our simulations show that using big constant step sizes leads to high variation and bad convergence behaviour. This is typical for stochastic optimization. We will now introduce an alternative to Contrastive Divergence training, leading to more robust optimization methods.

4.3 Pseudo-Likelihood

When using maximum likelihood, the intractabilities in the likelihood function and its gradient requires approximation of marginal probabilities with methods like the proposed MCMC with Gibbs sampler. Using an approximation of the objective function in each iteration of the optimization again causes problems, as mentioned above with contrastive divergence. Since the objective function is approximated it is likely to differ from the ground truth in each iteration step in a different way. This leads to jumpy gradients and broad epsilon bands around the minimum where no optimization is possible anymore even if long MCMC runs are used and the computation time exceeds acceptable bounds. Julian Besag [Bes75] proposed the use of pseudo-likelihood functions as objective function when optimizing highly multivariate models such as most graphical models. The idea is to use the conditional probability of the local marginal variables of a random field. In the case of classification on graphs, we use the probability of the label variables $\mathbf{Y}_v$ under the observations $\mathbf{X}$ and surrounding labels $\mathbf{Y}_{\mathrm{N}(v)\setminus v}$, where $v \in V$ denotes an arbitrary vertex in the underlying graph. Due to the Markovian nature of the CRF labels this leads to the factorization of the likelihood function.

$$L_{\mathcal{P}}(\boldsymbol{\theta}) = \prod_{v \in V} P_{\boldsymbol{\theta}}(y_v^{(0)} | \mathbf{x}_v^{(0)}, \mathbf{y}_{\setminus v}^{(0)}) = \prod_{v \in V} \frac{\exp\left(\boldsymbol{\theta}' \mathbf{f}(y_v^{(0)}, \mathbf{y}_{\setminus v}^{(0)}, \mathbf{x}_v^{(0)})\right)}{\sum_{y' \in \mathcal{Y}} \exp\left(\boldsymbol{\theta}' \mathbf{f}(y', \mathbf{y}_{\setminus v}^{(0)}, \mathbf{x}_v^{(0)})\right)}$$

The pseudo-loglikelihood function:

$$l_{\mathcal{P}}(\boldsymbol{\theta}; \mathbf{y}^{(0)}, \mathbf{x}^{(0)}) = \sum_{v \in V} \boldsymbol{\theta}' \mathbf{f}(\mathbf{y}_v^{(0)}, \mathbf{x}_v^{(0)}) - \ln \sum_{y' \in \mathcal{Y}} \exp\left(\boldsymbol{\theta}' \mathbf{f}(y', \mathbf{y}_{\setminus v}^{(0)}, \mathbf{x}_v^{(0)})\right) \tag{4.4}$$

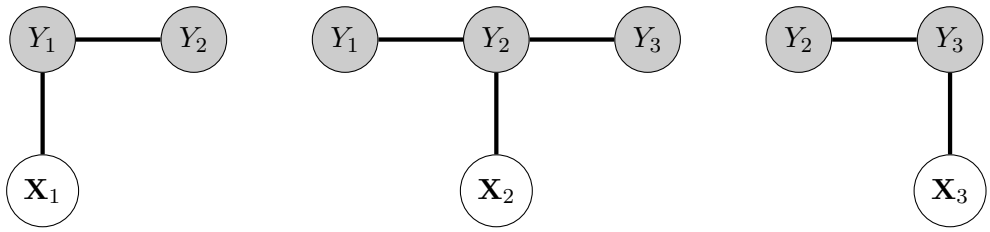
And the gradient becomes

$$\frac{\partial}{\partial \boldsymbol{\theta}_k} l_{\mathcal{P}}(\boldsymbol{\theta}; \mathbf{y}^{(0)}, \mathbf{x}^{(0)}) = \sum_{v \in V} f_k(\mathbf{y}_v^{(0)}, \mathbf{x}_v^{(0)}) - \sum_{v \in V} \sum_{{y'}_v} p({y'}_v | \mathbf{x}_v^{(0)}, \mathbf{y}_{\setminus v}^{(0)}) f_k({\mathbf{y}'}_v, \mathbf{x}^{(0)}) \tag{4.5}$$

$$= \sum_{v \in V} \left[f_k(\mathbf{y}_v^{(0)}, \mathbf{x}_v^{(0)}) - \sum_{{y'}_v} p({y'}_v | \mathbf{x}_v^{(0)}, \mathbf{y}_{\setminus v}^{(0)}) f_k({\mathbf{y}'}_v, \mathbf{x}^{(0)}) \right] \tag{4.6}$$

The probability term in the gradient of the log-likelihood $p({y'}_v|\mathbf{x}_v, \mathbf{y}_{\setminus v})$ can be calculated in constant time for every $v \in V$. Further, the summation term is only in one dimension and therefore $O(\mathcal{Y})$ instead of $O(\mathcal{Y}^{|V|})$, as in the original maximum likelihood or $O(\mathcal{Y}^{\max|\mathrm{N}(v)|})$ as in the simplified version 4.3

Figure 4.2: Pseudo-likelihood interpretation of a linear chain CRF - Node Splitting



4.3.1 Piecewise likelihood function

Besides using pseudolikelihood as approximation for the actual likelihood function there are some more approaches with different strengths and weaknesses. Sutton and McCallum [SM05] introduced the idea of piecewise likelihood functions. The idea is similar to the principle of pseudolikelihood. While the log pseudolikelihood is a sum over local conditional probabilities, piecewise likelihood sums over locally normalized Gibbs potentials.

$$l_{\mathrm{PW}}(\boldsymbol{\theta}) = \sum_{a \in \mathcal{F}} \log \frac{\Psi_a(\mathbf{y}_a, \mathbf{x}_a, \boldsymbol{\theta})}{\sum_{\mathbf{y}'} \Psi_a({\mathbf{y}'}_a, \mathbf{x}_a, \boldsymbol{\theta})}$$

In some scenarios this can be a better approximation to the likelihood than the pseudolikelihood. However we are focussing on cases where we are supplied with big datasets of complete information, where pseudolikelihood is a good approximation. In terms of computational complexity the piecewise likelihood is much more costly since the normalization term $\sum_{\mathbf{y}'}$ sums over all label configuration of the factor, which is of order $O(|\mathcal{Y}|^m)$ where m is the maximal vertex count per factor.
In a later paper Sutton and McCallum [SM07] enriched their method by combining piecewise likelihood and pseudolikelihood. They resolve the just mentioned computational cost

issue by using local pseudo likelihood estimations.

$$l_{\text{PWPL}} = \sum_{a \in F} \sum_{v \in a} \log \frac{\Psi_a(y_v, y_{\setminus v}, \mathbf{x}_a, \boldsymbol{\theta})}{\sum_{y'} \Psi_a(y', y_{\setminus v}, \mathbf{x}_a, \boldsymbol{\theta})}$$

In this paper [SM07] there is a brief empirical study comparing the introduced approximation methods, since all of them asymptotically lead to the same optimized model parameter. As a result the combined method is of highest efficiency whenever the training dataset is not big enough for pseudolikelihood to converge. Again, we focus on very big training datasets and therefore assume pseudolikelihood to be an optimal choice among the discussed likelihood approximations.

4.4 Regularization

Besides maximizing the log-likelihood function to find a suitable parameter $\boldsymbol{\theta}$ for the model we also investigate modifications of this objective function. On the one hand such modifications can help to avoid overfitting as well as simplifying the model by sparsening $\boldsymbol{\theta}$. On the other hand a we can introduce penalty terms to ensure strict convexity of the objective function. A commonly used approach is to construct an objective function by adding a penalty term to the log-likelihood function. In our work we want to compare three popular approaches (L1, L2 and elastic net penalty). Regularization via penalty terms usually implies a penalty that only depends on the model parameter $\boldsymbol{\theta}$.

4.4.1 L1 - penalty

Using the L1 norm of the model parameter as penalty term has become very popular with a variety of models. The initial formulation, its properties in the context of linear regression models is described in an updated paper by Tibshirani [Tib11]. The basic idea is the following:
Let $l(\cdot)$ denote the log-likelihood function, then we formulate the L1-penalized likelihood as:

$$g(\boldsymbol{\theta}) = l(\boldsymbol{\theta}) + \lambda \sum_k (|\theta_k|) \tag{4.7}$$

with $\lambda > 0$ beeing an adjustable parameter determining the strength of the regularization term. Often we look at the regularized likelihood as the Lagrangian formulation of the minimization problem:

$$\boldsymbol{\theta}^* = \operatorname{argmin} l(\boldsymbol{\theta}) \quad \text{under the contraint} \quad \sum_k (|\theta_k|) < t \tag{4.8}$$

for preselected $t > 0$. For linear problems there exist effective inductive algorithm for solving the minimization problem and creating a full regularization path. Implementations

of the LASSO formulation by Tibshirani [Tib11] used a stepwise forward induction based on the correlation of the covariates and the residuals. A major step forward was the LARS algorithm by Efron [EHJT04] which also resides on correlation between residuals and the covariates. A short summary and overview over the L1 variants can be found in [Tib11]. Due to the Markovian nature of the CRF it is not possible to determine the residuals each iteration. This requires a full inference step (i.e. via MCMC, as in chapter 3), which is computationally expensive. This renders the classic regularization and shrinkage algorithms for linear models intractable or at least inefficient for CRF problems on big graphs. Although these algorithms cannot directly be applied to the CRF setting we can take advantage of the shrinkage and variable selection behaviour of the L1 term. The main benefit from using L1 for model shrinking and variable selection is, that the penalty term is sharp (and not differentiable) around the origin in each dimension. This results in a regularization that achieves exact zeros as entries in $\boldsymbol{\theta}$ since the gradient in of the penalty term is constant. The Baysian interpretation of the L1-regularized log-likelihood is that this solves not the original ML problem but a maximum a posteriori with an exponential prior distribution [GS03].

4.4.2 L2 and elastic net

Analogously to the L1 penalty we can use an L2 term to the objective function for regularization and regard the penalized likelihood minimization as solving the problem:

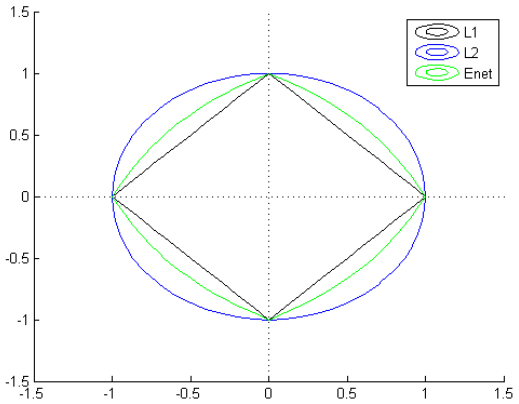
$$\boldsymbol{\theta}^* = \operatorname{argmin} l(\boldsymbol{\theta}) \quad \text{under the contraint} \quad \sum_k (\theta_k^2) < t \tag{4.9}$$

for preselected $t > 0$. Especially in the linear regression scenario this regularization, called Ridge Regression, is widely used for it has two benefits over the L1 penalization. It encourages the grouping of variables, which allows for a better interpretation. Further it is very useful in cases where $\dim(\boldsymbol{\theta}) > |V|$ or as usually formulated in standard literature '$p > n$' problems. A third, in our case important feature of L2 regularization is the fact, that the L2 norm is strictly convex. Analogously to the L1 regularization this optimization problem can be viewed as a MAP solution, but here with a Gaussian prior distribution on $\boldsymbol{\theta}$. The original CRF log-likelihood is not necessarily strictly convex. Using L2 regularization can be used to receive a strictly convex objective function, since the penalty term is strictly convex. However the L2 term has a flat gradient when approaching zero.

$$\frac{\partial}{\partial \theta_l} \sum_k (\theta_k^2) = 2\theta_l \tag{4.10}$$

This usually results in a very slow achievement of exact zero entries in the model parameter and is therefore not well suited for variable selection. In their paper [ZH05] discuss the combination of L1 and L2 regularization for regression models. Their approach is called the elastic-net. In their paper they also present an algorithm for inducing model parameters in the elastic net setting, similar to LARS for the L1 penalty. However this again is not

Figure 4.3: Penalty terms of $\boldsymbol{\theta} \in \mathbb{R}^2$ with contourline at 1



directly applicable to the CRF setting. We therefore just borrow the idea of combining L1 and L2, which gives us a penalization with the following benefits:

- Shrinkage - Model parameters are reduced
- Variable Selection - Irrelevant model parameters become exact zeros
- Convexity - The penalization term is strictly convex and therefore the objective function is too

4.4.3 Thresholding

When we use penalty terms to receive a sparse model, we can specify the iterative optimization algorithm to use a consequently reduced version of the model. This means we combine the model selection idea from stepwise regression with the optimization procedures that are described in detail in chapter 5. A straight forward implementation for an iterative optimization algorithm with thresholding is to define a threshold value $\epsilon > 0$ and remove a feature, or generally speaking, a dimension of the parameter space, whenever $\boldsymbol{\theta}_i < \epsilon$. There are some practical issues that have to be handled:

The L1-gradient is a step function in each direction with a singularity at zero. In practical applications however values of exact zero are often not achieved which calls for the definition of an $\epsilon > 0, \epsilon \approx 0$, which serves as numerical boundary for the area that is interpreted as zero. Once we specify this epsilon in the derivative computation we have to keep in mind that values underneath have to be consequently be interpreted as zero. Hence the ϵ used for thresholding and for the gradient should be the same.

Further we have to keep in mind that values of nonzero are always to regarded as beeing relevant for the model. This is a great benefit of using the L1 or elastic net penalty. In

order to sort the dimensions of the model parameter in terms of importance, we can use their absolute values. However this requires a full normalization of the parameters dependent on the observation of the covariates $\mathbf{x}$ as described in 2.5.2. Whenever this is not possible or efficient, we have to consider that the model contribution is not directly given by the model parameter, but by the Gibbs potential:

$$E(\cdot) = \boldsymbol{\theta}'\mathbf{f}(\cdot)$$

Hence, if a penalty term is to be interpreted as an indicator for the strengths of a certain parameter $\boldsymbol{\theta}^*$ it is necessary to take the feature function definition $\mathbf{f}$ into account.

4.5 Summary

The training of CRFs using classic maximum likelihood approaches is an intractable task when dealing with big graphs and big parameter spaces. This mainly stems from the fact that the normalization term $Z(\mathbf{x})$ is a sum over all possible states of the target variable in all nodes of the graph. Approximating this term in each optimization iteration by using MCMC inference steps is not efficient and leads to jittery objective functions. The approach of minimizing contrastive divergence shows that also very few MCMC iterations can be used to solve the ML equivalent problem of minimizing the contrastive divergence between the current state of the model. Using the the MCMC sample information together with their Gibbs energy terms leads to a contrastive divergence method which is almost independent of the amount of MCMC samples. Although this approach enables us to use contrastive divergence with a single MCMC run, the problem remains, that the objective function is approximated each iteration anew. For linesearch optimization this is a hard problem, since there is no deterministic way to find a suitable stepsize which ensures efficient convergence. A more robust method is to use approximate the likelihood function by using the conditional probabilities and optimizing the pseudo-likelihood function. We will focus on using the pseudo-likelihood approach when we turn to the numeric optimization in the next chapter.

5 Optimization

This chapter deals with the numerical optimization that is used for the training of conditional random fields. In the last chapter we introduced different approaches for model training and we focus now on the optimization of the corresponding objective functions. In section 5.1 we show that in the maximum likelihood training case, the objective function is convex and we can therefore make use of standard optimization techniques. We introduce widely used line search approaches in section 5.2, explain the application related difficulties and introduce ways to cope with them. In section 5.3 we show stochastic optimization methods that are commonly used in the CRF context. One specific stochastic optimization approach, stochastic average gradient descent, is explained in 5.4. In the same section we also introduce variations of this approach. Afterwards we present a new enhanced variant of the stochastic average gradient method.

5.1 Objective function

In the context of CRF training via maximum likelihood or as introduced in section 4.3 via Pseudo-Likelihood we optimize strictly convex objective functions. Before dealing with the optimization procedures themselves we start by looking at the nature of the objective function.

5.1.1 Convexity of pseudo-likelihood

As stated above, training a CRF via minimizing the negative log-pseudolikelihood function is a convex optimization problem. The convexity of the objective function can be shown several ways. One possibility is to exploit the fact, that log-exp-sum like functions are always convex. Another way which also yields some more insights is to show that the hessian is positive definite. Therefore we compute the Hessian of the negative pseudo log-pseudolikelihood. Given a training dataset $S = (\mathbf{y}, \mathbf{x}) = (y_v, \mathbf{x}_v)_{v \in V}$ the log-pseudolikelihood is:

$$l_{\mathcal{P}}((\boldsymbol{\theta}; \mathbf{y}, \mathbf{x}) = \sum_{v \in V} \boldsymbol{\theta}' \mathbf{f}(\mathbf{y}_v, \mathbf{x}_v) - \ln \sum_{y' \in \mathcal{Y}} \exp\left(\boldsymbol{\theta}' \mathbf{f}(y', \mathbf{y}_{\setminus v}, \mathbf{x}_v)\right) \tag{5.1}$$

The first derivative:

$$\frac{\partial}{\partial \boldsymbol{\theta}_k} l(\boldsymbol{\theta}; \mathbf{y}, \mathbf{x}) = \sum_{v \in V} \left[f_k(\mathbf{y}_v, \mathbf{x}_v) - \sum_{y'} \frac{\exp\left(\boldsymbol{\theta}' \mathbf{f}(y', \mathbf{y}_{\setminus v}, \mathbf{x}_v)\right)}{\sum_{y'' \in \mathcal{Y}} \exp\left(\boldsymbol{\theta}' \mathbf{f}(y'', \mathbf{y}_{\setminus v}, \mathbf{x}_v)\right)} f_k(y', \mathbf{y}_{\setminus v}, \mathbf{x}) \right] \tag{5.2}$$

The hessian at (k, l):

$$\frac{\partial^2}{\partial \boldsymbol{\theta}_k \boldsymbol{\theta}_l} l(\boldsymbol{\theta}; \mathbf{y}, \mathbf{x}) \quad = \quad \sum_{v \in V} \frac{\partial}{\partial \boldsymbol{\theta}_l} \left[- \sum_{y'} \frac{\exp\left(\boldsymbol{\theta}' \mathbf{f}(y', \mathbf{y}_{\setminus v}, \mathbf{x}_v)\right)}{\sum_{y'' \in \mathcal{Y}} \exp\left(\boldsymbol{\theta}' \mathbf{f}(y'', \mathbf{y}_{\setminus v}, \mathbf{x}_v)\right)} f_k(y', \mathbf{y}_{\setminus v}, \mathbf{x}) \right] \tag{5.3}$$

Now for a fixed vertex v and for the sake of readability, omitting the dependency of $\mathbf{y}_{\setminus v}$ and $\mathbf{x}_v$ we get:

$$\begin{aligned}
&- \frac{\partial^2}{\partial \boldsymbol{\theta}_k \boldsymbol{\theta}_l} l_v(\boldsymbol{\theta}; \mathbf{y}, \mathbf{x}^{(0)}) = \frac{\partial}{\partial \boldsymbol{\theta}_l} \left[\sum_{y'} \frac{\exp\left(\boldsymbol{\theta}' \mathbf{f}(y')\right)}{\sum_{y' \in \mathcal{Y}} \exp\left(\boldsymbol{\theta}' \mathbf{f}(y')\right)} f_k(y') \right] \\
&= \frac{\partial}{\partial \boldsymbol{\theta}_l} \frac{\sum_{y'} f_k(y') \exp\left(\boldsymbol{\theta}' \mathbf{f}(y')\right)}{\sum_{y' \in \mathcal{Y}} \exp\left(\boldsymbol{\theta}' \mathbf{f}(y')\right)} \\
&= \frac{\left[\sum_{y'} f_k(y') f_l(y') \exp\left(\boldsymbol{\theta}' \mathbf{f}(y')\right)\right] \left[\sum_{y' \in \mathcal{Y}} \exp\left(\boldsymbol{\theta}' \mathbf{f}(y')\right)\right]}{\left[\sum_{y' \in \mathcal{Y}} \exp\left(\boldsymbol{\theta}' \mathbf{f}(y')\right)\right]^2} \\
&\qquad - \frac{\left[\sum_{y'} f_k(y') \exp\left(\boldsymbol{\theta}' \mathbf{f}(y')\right)\right] \left[\sum_{y'} f_l(y') \exp\left(\boldsymbol{\theta}' \mathbf{f}(y')\right)\right]}{\left[\sum_{y' \in \mathcal{Y}} \exp\left(\boldsymbol{\theta}' \mathbf{f}(y')\right)\right]^2} \\
&= \sum_{y'} f_k(y') f_l(y') \frac{\exp\left(\boldsymbol{\theta}' \mathbf{f}(y')\right)}{\sum_{y'' \in \mathcal{Y}} \exp\left(\boldsymbol{\theta}' \mathbf{f}(y'')\right)} \\
&\qquad - \left[\sum_{y' \in \mathcal{Y}} f_k(y') \frac{\exp\left(\boldsymbol{\theta}' \mathbf{f}(y')\right)}{\sum_{y'' \in \mathcal{Y}} \exp\left(\boldsymbol{\theta}' \mathbf{f}(y'')\right)} \right] \left[\sum_{y' \in \mathcal{Y}} f_l(y') \frac{\exp\left(\boldsymbol{\theta}' \mathbf{f}(y')\right)}{\sum_{y'' \in \mathcal{Y}} \exp\left(\boldsymbol{\theta}' \mathbf{f}(y'')\right)} \right] \\
&= \sum_{y'} f_k(y') f_l(y') p(y') - \left[\sum_{y'} f_k(y') p(y') \right] \left[\sum_{y'} f_l(y') p(y') \right] \\
&= \mathbf{E}\left[f_k(Y) f_l(Y) \right]_{\boldsymbol{\theta}, v} - \mathbf{E}\left[f_k(Y) \right]_{\boldsymbol{\theta}, v} \mathbf{E}\left[f_l(Y) \right]_{\boldsymbol{\theta}, v} = \mathrm{Cov}_{\boldsymbol{\theta}, \mathbf{x}_v}(f_k(Y), f_l(Y))
\end{aligned} \tag{5.4}$$

with $\mathbf{E}\left[\cdot\right]_{\boldsymbol{\theta}, \mathbf{x}_v}$ being the expected value for the distribution of Y with

$$p(Y = y) = \frac{\exp(\boldsymbol{\theta}' \mathbf{f}(\mathbf{x}_v, y))}{\sum_{y''} \exp(\boldsymbol{\theta}' \mathbf{f}(\mathbf{x}_v, y''))}$$

This yields that the Hessian of the negative log likelihood function can be written as sum of covariance matrices $\sum_v \mathrm{Cov}(\mathbf{f}(\mathbf{x}_v, Y))$ and is therefore positive semi-definite.

5.1.2 Observed Fisher information

A valuable characteristic of the objective function is the observed fisher information. It grants us an insight about how sharp the objective function is and therefore how robust towards the optimization parameters. We are interested in the fisher information of the maximum-pseudolikelihood approach. In this case the observed fisher information is given by the hessian of the log-pseudolikelihood, as calculated above:

$$\frac{\partial^2}{\partial \boldsymbol{\theta}_k \boldsymbol{\theta}_l} l(\boldsymbol{\theta}; \mathbf{y}, \mathbf{x}) = \sum_v \mathrm{Cov}_{\boldsymbol{\theta}, \mathbf{x}_v}(f_k(\mathbf{x}_v, Y), f_l(\mathbf{x}_v, Y))$$

The way we choose the CRF's feature functions and the fact that we use feature normalization leads to small covariances and a flat Hessian near the optimum. Although there is no analytical boundary for the absolute value of the covariances that shows a degree of flatness, we can see, that the Fisher information is small if:

- Feature functions are normalized (which we assume) and therefore limited in range
- The stochastic dependency between observations is small
- The underlying probability $P^Y(\cdot|\mathbf{y}\backslash_v, \mathbf{x}_v)$ is of low entropy, i.e. has distinct peaks.

In practical applications we observe that these criteria are usually met, which leads to a low Fisher information.

5.2 Line search algorithms

In order to find the maximum of the likelihood function we use an iterative optimization algorithm. All possible objective functions defined in chapter 4 are nonlinear and leave us to numerically solving procedures. There is a wide variety of methods available and we will discuss which algorithms are best suited in case of typical CRFs and especially in cases that are related to the road-quality oriented scenario. A linesearch optimization algorithm seperates the determination of the optimization direction p in parameter space and the step size α. Let the model parameter be $\boldsymbol{\theta}$, then a prototypical line search step in the k-th iteration is:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \alpha_k \cdot p_k$$

In general we can divide the field of available linesearch algorithms into a set of algorithms using second order information, algorithms that approximate this information and algorithms not using second order information at all. This categorization is important to the optimization of likelihood functions of CRFs since this is usually a high dimensional problem. Second order algorithms have to determine the objective functions Hessian with regard to the parameters. There many use cases, for example in the field of natural language processing with huge parameter spaces, which make even storing the second order information impossible. The same holds true for our target application, the road quality

measure extrapolation. Since calculating the Hessian is inefficient for such sizes, we will now compare Semi-Newton and gradient-based methods

5.2.1 Semi-Newton methods

Before introducing semi-Newton methods we briefly restate the typical procedure of Newton methods. These methods use the gradient ∇g as well as the Hessian H of the objective function $\mathbf{g}$. The iterative optimization is perfomed by setting:

$$\boldsymbol{\theta}_{n+1} = \boldsymbol{\theta}_n - \alpha \cdot [Hf(\boldsymbol{\theta}_n)]^{-1} \nabla g(\boldsymbol{\theta}_n),\ n \geq 0$$

This requires calculating the Hessian in each iteration, which is of complexity $O(\dim\boldsymbol{\theta}^2)$. In typical classification tasks this becomes inefficient in space and time. Choosing a good stepsize α ensures convergency (for example fulfilling Wolfe conditions 5.2.1). Calculation of the inverse is also very expensive. Therefore one directly calculates $\mathbf{p}_n := [Hf(\boldsymbol{\theta}_n)]^{-1} \nabla g(\boldsymbol{\theta}_n)$ by solving the linear system:

$$[Hf(\boldsymbol{\theta}_n)]\mathbf{p}_n = \nabla g(\boldsymbol{\theta}_n)$$

Semi Newton methods approximate the hessian H by finite differences. In such approaches we are looking for an approximation matrix B which holds:

$$\nabla g(\boldsymbol{\theta}_k + \Delta\boldsymbol{\theta}) = \nabla g(\boldsymbol{\theta}_k) + B\,\Delta\boldsymbol{\theta}$$

This is an underdefined problem and can be solved with a variety of approaches.

BFGS

A very popular approach of approximating the Hessian is used by the BFGS ([Bro70],[Gol70],[Sha70]) algorithm. It uses finite differences of the gradient to approximate the Hessian in a single dimension in every iteration.

$$B_{k+1} = B_k + \frac{y_k y_k^T}{y_k^T \Delta\boldsymbol{\theta}_k} - \frac{B_k \Delta\boldsymbol{\theta}_k (B_k \Delta\boldsymbol{\theta}_k)^T}{\Delta\boldsymbol{\theta}_k^T B_k\, \Delta x_k} \quad \text{with} \quad y_k := \nabla g(\boldsymbol{\theta}_{k+1}) - \nabla g(\boldsymbol{\theta}_k)$$

The inverse can also be defined iteratively by applying the Sherman–Morrison formula [SM50]:

$$H_{k+1} = \left(I - \frac{y_k \Delta\boldsymbol{\theta}_k^T}{y_k^T \Delta\boldsymbol{\theta}_k}\right)^T H_k \left(I - \frac{y_k \Delta\boldsymbol{\theta}_k^T}{y_k^T \Delta\boldsymbol{\theta}_k}\right) + \frac{\Delta\boldsymbol{\theta}_k \Delta x_k^T}{y_k^T\, \Delta\boldsymbol{\theta}_k}$$

This Hessian approximation is of constant computational complexity for each iteration.

L-BFGS

Whenever we are dealing with high dimensional parameter spaces, storing the approximation of a hessian as a dense $|\dim\boldsymbol{\theta}| \times |\dim\boldsymbol{\theta}|$ matrix is very costly. The limited memory BFGS (L-BFGS) only stores vector representations of the hessian approximation matrix B. This leads to a linear space complexity. The pseudo code of the LBFGS is given in 5.2.

Algorithm 5.1 Broyden-Fletcher-Goldfarb-Shannon Algorithm for finding local minima

t := 0
$\mathbf{B}_0 = \mathbf{I}$
while $\|\nabla g(\boldsymbol{\theta}_t)\| > \epsilon$ **do**
    (a) $\mathbf{p}_t = -\mathbf{B}_t \nabla g(\boldsymbol{\theta}_t)$
    (b) $\eta_t = \text{linemin}(g, \boldsymbol{\theta}_t, \mathbf{p}_t)$
    (c) $\mathbf{s}_t = \eta_t \mathbf{p}_t$
    (d) $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \mathbf{s}_t$
    (e) $\mathbf{y}_t = \nabla g(\boldsymbol{\theta}_{t+1}) - \nabla g(\boldsymbol{\theta}_t)$
    (f) if $t = 0 : \mathbf{B}_t := \frac{\mathbf{s}_t' \mathbf{y}_t}{\mathbf{y}_t' \mathbf{y}_t} \mathbf{I}$
    (g) $\rho_t = (\mathbf{s}_t' \mathbf{y}_t)^{-1}$
    (h) $\mathbf{B}_{t+1} = (\mathbf{I} - \rho_t \mathbf{s}_t \mathbf{y}_t') \mathbf{B}_t (\mathbf{I} - \rho_t \mathbf{y}_t \mathbf{s}_t') + \rho_t \mathbf{s}_t \mathbf{s}_t'$
    (i) $t := t + 1$
end while
return $\boldsymbol{\theta}_t$

Algorithm 5.2 L-BFGS algorithm for finding local minima

t := 0
while $\|\nabla g(\boldsymbol{\theta}_t)\| > \epsilon$ **do**
    (a1) $\mathbf{p}_t = -\nabla g(\boldsymbol{\theta}_t)$
    for $i := 1, 2, \ldots, \min(t, m)$ **do**
        (a2) $\alpha_i = \frac{\mathbf{s}_{t-i}' \mathbf{p}_t}{\mathbf{s}_{t-i}' \mathbf{y}_{t-i}}$
        (a3) $\mathbf{p}_t := p_t - \alpha_i \mathbf{y}_{t-i}$
    end for
    (a4) if $t > 0 : \mathbf{p}_t := \frac{\mathbf{s}_{t-1}' \mathbf{y}_{t-1}}{\mathbf{y}_{t-1}' \mathbf{y}_{t-1}} \mathbf{p}_t$
    for $i := \min(t, m), \ldots, 2, 1$ **do**
        (a5) $\beta = \frac{\mathbf{y}_{t-i}' \mathbf{p}_t}{\mathbf{s}_{t-i}' \mathbf{y}_{t-i}}$
        (a6) $\mathbf{p}_t := \mathbf{p}_t + (\alpha_i - \beta) \mathbf{s}_{t-i}$
    end for
    (b) $\eta_t = \text{linemin}(f, \boldsymbol{\theta}_t, \mathbf{p}_t)$
    (c) $\mathbf{s}_t = \eta_t \mathbf{p}_t$
    (d) $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \mathbf{s}_t$
    (e) $\mathbf{y}_t = \nabla g(\boldsymbol{\theta}_{t+1}) - \nabla g(\boldsymbol{\theta}_t)$
    (i) $t := t + 1$
end while
return $\boldsymbol{\theta}_t$

5.2.2 Gradient descent

The general idea is to solve the minimization problem

$$x^* = \text{argmin}_x f(x)$$

by finding the root of the gradient $\nabla g(\mathbf{x})$ via fix-point iteration. This iterative process is shown in algorithm 5.3. Since it only requires the gradient to exist, it is applicable to a big spectrum of objective functions. The drawback of this approach is essentially the fact that the gradient is likely to be a less accurate choice for a correct descent direction, compared to the second order approximations of Newton and Semi-Newton methods.

Algorithm 5.3 Simple Gradient Descent with fixed step size α

```
1: while not converged do
2:     x = x − α · ∇g(x)
3: end while
```

Algorithm 5.4 Gradient Descent with variable step size α

```
1: while not converged do
2:     g = ∇g(x)
3:     y = x + α · g/‖g‖
4:     if f(y) > f(x) then
5:         while f(y) > f(x) do
6:             α = α · 0.5
7:             y = x − α · g/‖g‖
8:         end while
9:     else
10:        α = α · 1.2
11:    end if
12: end while
```

5.2.3 Step size

As seen in algorithm 5.4 we use the normalized gradient as the direction and determine the length α in a separate step. This has to be done since the original length of the gradient is not applicable. The gradient's original scale is oriented in the range of the objective function, which is usually completely different from the parameter space, where we want to apply the changes to. The idea is now to iteratively determine a suitable step size. There are different sets of conditions that ensure the convergence of the algorithm by setting restrictions to the step size. We will shortly discuss the Wolfe conditions and strong Wolfe conditions ([Wol71]). Each of these rules can be used for backtracking algorithms to find good step sizes in line search algorithms. Both conditions ensure the sufficient decrease of the objective function and the convergence of its gradient to zero.

Definition 5.2.1. *Wolfe conditions Given an objective function $g(\boldsymbol{\theta})$, its gradient $\nabla g(\boldsymbol{\theta})$, a descent direction $\mathbf{d}$, then a stepsize $\alpha \in \mathbb{R}_+$ is said to hold the armijo rule iff for a given $c_1 > 0$ and $c_2 > 0$ the following inequalities are true:*

1. $g(\boldsymbol{\theta} + \alpha \mathbf{d}) \leq g(\boldsymbol{\theta}) + c_1 \cdot \alpha \cdot \mathbf{d}' \nabla g(\boldsymbol{\theta})$
2. $\mathbf{d}' \nabla g(\boldsymbol{\theta} + \alpha \mathbf{d}) \geq c_2 \cdot \mathbf{d}' \nabla g(\boldsymbol{\theta})$

Definition 5.2.2. *Strong Wolfe conditions Given an objective function $g(\boldsymbol{\theta})$, its gradient $\nabla g(\boldsymbol{\theta})$, a descent direction $\mathbf{d}$, then a stepsize $\alpha \in \mathbb{R}_+$ is said to hold the armijo rule iff for a given $c_1 > 0$ and $c_2 > 0$ the following inequalities are true:*

1. $g(\boldsymbol{\theta} + \alpha \mathbf{d}) \leq g(\boldsymbol{\theta}) + c_1 \cdot \alpha \cdot \mathbf{d}' \nabla g(\boldsymbol{\theta})$
2. $|\mathbf{d}' \nabla g(\boldsymbol{\theta} + \alpha d)| \leq c_2 \cdot |\mathbf{d}' \nabla g(\boldsymbol{\theta})|$

5.3 Stochastic approaches

The introduced methods require that evaluating the objective function g as well as calculating its gradient ∇g is not too costly. We will now focus on problems and in particular classification problems for which the objective function does not suffice that. In the case of maximum pseudolikelihood on CRFs, gradient and function evaluations are of complexity $O(|V| \cdot |\boldsymbol{\theta}|)$. In the motivation chapter we gave an overview over the typical dimensions we are dealing with when training a Conditional Random Field that is supposed to model a countries road network.
The log-pseudolikelihood of a CRF can be represented as sum over the vertices of the underlying graphical structure. Whenever an objective function is of such a form, stochastic optimization can be applied straight forward. One selects different subsets of vertices $V_t \subset V$ in each iteration and optimizes the likelihood that just takes into account the elements within the subset. Defining the amount of vertices that are taken into account each iteration determines the accuracy of the approximation of the log-likelihood but also determines the computation time for the objective function and its gradient.
Since we are dealing with iterative procedures the stochastic approximations of the objective functions can be indexed by the iteration t and are defined by:

$$g(\boldsymbol{\theta}) = \sum_{v \in V} g_v(\boldsymbol{\theta}) \qquad \rightarrow \qquad g_t(\boldsymbol{\theta}) := \sum_{v \in V_t} g_v(\boldsymbol{\theta})$$

with V_t being the random subset of vertices in iteration t and $f_v(\boldsymbol{\theta})$ being the contribution of vertex v to the objective function.

5.3.1 Stochastic Gradient Descent

The algorithm differs from the deterministic gradient descend algorithm 5.5 in the single aspect of selecting random subsets of vertices $V_t \subseteq V$. In the case of the log-pseudolikelihood the objective function is in the form of a sum over vertices $\sum_{v \in V}$, which requires only the replacement of the index set by the subset V_t. When we compare convergence behavior of

a stochatic gradient method with a standard gradient descent we note some major differences when using fixed step sizes. First it to say that a comparison between deterministic and stochastic gradient descent is highly dependent on the fixed stepsize or the stepsize strategy. As mentioned above, in stochastic approaches, standard stepsize criteria, like the Wolfe criteria are not guaranteeing convergence and are generally not applicable. Using fixed step sizes encourages to compare the convergence result on a 'per step' and therefore 'per iteration' base. The stochastic variants show a degree of noise at later iterations which is typical for stochastic approaches and fixed step sizes.

Algorithm 5.5 Stochastic Gradient Descent with fixed step size α

```
1: while not converged do
2:     Select random batch of vertices V_t ⊆ V
3:     x = x − α · ∇f_t(x)
4: end while
```

5.3.2 Step size decay

One way to choose a step size for stochastic algorithms directly derives from the characteristics of the parameter $\boldsymbol{\theta}$ viewed as a stochastic

Definition 5.3.1 (Robbins-Monro-Process)**.** *Let Y_θ be a random variable over $\mathbb{R}$ and $M(\theta) = E(Y_\theta)$. Further there must exist a unique $\theta^* \in \mathbb{R}$ with $M(\theta^*) = 0$, the process defined by*

$$\theta_{n+1} = \theta_n + a_n Y_\theta$$

is called a Robbins-Monro-Process, where $a_n > 0 \forall n \in \mathbb{N}$ and a arbitrary starting point $\theta_0 \in \mathbb{R}$.

In [RM51] is shown that a Robbins-Monro-Process converges with $n \to \infty$

$$\theta_n \to \theta^* \quad \text{in} \quad L^2$$

and therefore also in probability if the following conditions hold:

1. $M(\theta)$ is non-decreasing
2. $\frac{\partial \theta_i}{\partial M}(\theta) > 0$
3. Y is essentially uniformly bound
4. The stepsize series a_n holds:

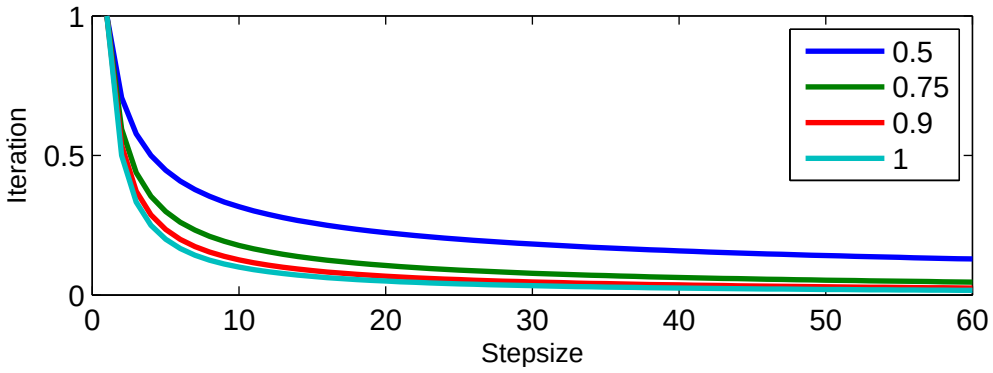
$$\sum_{n=0}^{\infty} a_n = \infty \quad \text{and} \quad \sum_{n=0}^{\infty} a_n^2 < \infty$$

In the case of stochastic optimization via line search we observe that the iterative process of finding a root of the gradient is in the form of a multivariate Robbins-Monro-Process. Therefore the conditions for the stepsize a_n, which are sometimes simply called Robbins-Monro conditions are essential when defining a step size strategy.
Popular strategies that hold these conditions are:

$$a_n = \frac{1}{n^\alpha}$$

where $0.5 < \alpha \leq 1$ is a tuning parameter which controls the decay rate of the stepsize.

Figure 5.1: Decay function for different α



5.4 Stochastic Average Gradient

5.4.1 Basic method

A major enhancement to SGD was achieved by Roux et al. [RSB12] who introduced a stochastic descend algorithm of exponential convergence rate. This method is called stochastic average gradient. Each iteration, the algorithm uses gradient information from the current random subsample of V but also incorporates the gradient information of past iterations. Given an objective function in the form of a sum of contributions like the following:

$$g(\boldsymbol{\theta}) = \sum_v g_v(\boldsymbol{\theta}) \quad \text{and} \quad \nabla g(\boldsymbol{\theta}) = \sum_v \nabla g_v(\boldsymbol{\theta}) \tag{5.5}$$

Since the pseudo likelihood function can be written as sum over the individual vertices, the same holds true for the gradient. Adding penalty terms does not change this characteristics. The basic principle is to update the vertex-wise gradient contributions only for

a subset of vertices. Given such a random subset $V_t \subset V$ in the t-th iteration the update vector p_t is calculated via:

$$p_t \propto \nabla g(\boldsymbol{\theta}_t) = \sum_v w_v^{(t)} \quad \text{with} \quad w_{t,v} = \begin{cases} \nabla g_v(\boldsymbol{\theta}) & , v \in V_t \\ w_{t-1,v} & , v \notin V_t \end{cases}$$

The stochastic average gradient approach assumes that the gradient or at least its direction is not likely to change immensely every iteration. This behaviour can be enforced by using very small stepsizes. The pseudo code (5.4.1) demonstrates the use of contributions $\mathbf{w}_v^i := \nabla g_v(\boldsymbol{\theta}^i)$ of the vertex v to the gradient in iteration i. The constant W used in the pseudo code can be used to limit the number of contributions.

Algorithm 5.6 Stochastic Average Gradient with mini-batches

1: Initialize $w_v^0 \quad \forall v \in V$
2: **while** not converged **do**
3: Determine a random sub sample $V_s \subset V$
4: **for** j = 1 to W **do**
5: $G = \sum_{v \in V} w_v^i(\boldsymbol{\theta})$
6: With $\mathbf{w}_v^i = \begin{cases} \nabla g_v(\boldsymbol{\theta}) & if \quad v \in V_i \\ w_v^{i-1} & \text{else} \end{cases}$
7: Determine stepsize α [using V or V_i]
8: $\mathbf{y} = \mathbf{x} - \alpha \cdot \frac{G}{\|G\|}$
9: **end for**
10: **end while**

5.4.2 SAG in the context of CRFs

The SAG algorithm as introduced is applicable to all kinds of problems where the objective function is convex and can be written in the form of a sum as in 5.5. In the case of CRFs with the log-pseudolikelihood as objective function for model training the set over which is being summed is the set of vertices of the graph V. We will now focus on properties, issues and solutions for the SAG in this context and especially for big graphs and big parameter spaces.

Stepsize

The exponential convergence rate of the SAG algorithm is only guaranteed for rather small stepsizes. The step size boundary is discussed in the paper by Roux et al [RSB12] and without any further restrictions is given by:

$$\alpha \leq \frac{1}{2nL}$$

where L denotes the Lipschitz constant of the individual summands of the objective function. All other boundaries given by Roux et al[RSB12] that result from more restrictive model configurations are also dependent on the Lipschitz constant. In the case of the pseudolikelihood in the CRF context as objective function we get:

$$l_{\mathrm{PL}} = \log\left(\frac{1}{Z}\prod_v \exp(\boldsymbol{\theta}'\mathbf{f}(\mathbf{x}_v, y_v, \mathbf{y}_{\setminus v}))\right) = \sum_v g_v(\boldsymbol{\theta})$$

$$g_v(\boldsymbol{\theta}) := \boldsymbol{\theta}'\mathbf{f}(\mathbf{x}_v, y_v, \mathbf{y}_{\setminus v}) - \log\left(\sum_{y'\in\mathcal{Y}} \exp(\boldsymbol{\theta}'\mathbf{f}(\mathbf{x}_v, y_v, \mathbf{y}_{\setminus v}))\right)$$

The relevant Lipschitz constant is the one of the vertex wise objective $g_v(\cdot)$:

$$\begin{aligned}
|g_v(\boldsymbol{\theta}_1) - g_v(\boldsymbol{\theta}_2)| &= \left|(\boldsymbol{\theta}_1 - \boldsymbol{\theta}_2)'\mathbf{f}(\mathbf{x}_v, y_v, \mathbf{y}_{\setminus v}))) - \log\left(\frac{\sum_{y'\in\mathcal{Y}} \exp(\boldsymbol{\theta}_1'\mathbf{f}(\mathbf{x}_v, y', \mathbf{y}_{\setminus v})))}{\sum_{y'\in\mathcal{Y}} \exp(\boldsymbol{\theta}_2'\mathbf{f}(\mathbf{x}_v, y', \mathbf{y}_{\setminus v})))}\right)\right| \\
&\leq \left|(\boldsymbol{\theta}_1 - \boldsymbol{\theta}_2)'\mathbf{f}(\mathbf{x}_v, y_v, \mathbf{y}_{\setminus v})))\right| + \left|\log\left(\frac{\sum_{y'\in\mathcal{Y}} \exp(\boldsymbol{\theta}_1'\mathbf{f}(\mathbf{x}_v, y', \mathbf{y}_{\setminus v})))}{\sum_{y'\in\mathcal{Y}} \exp(\boldsymbol{\theta}_2'\mathbf{f}(\mathbf{x}_v, y', \mathbf{y}_{\setminus v})))}\right)\right|
\end{aligned}$$

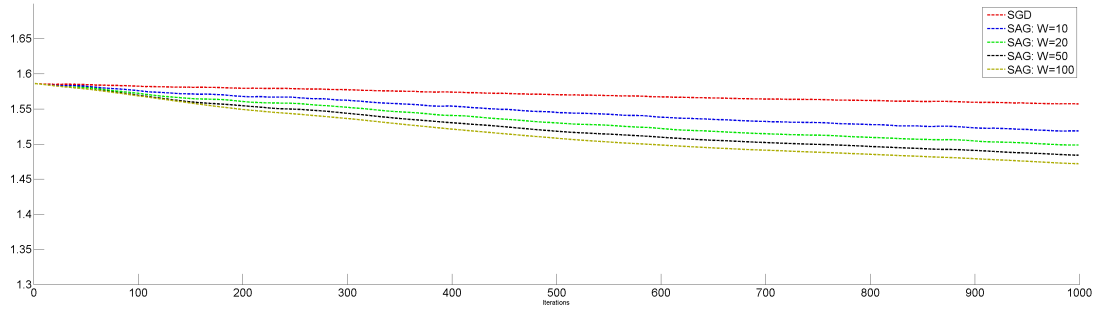
The following inequality can be derived from Jensen's inequality:

$$\sum_{i=1}^n a_i \log\frac{a_i}{b_i} \geq \sum_{i=1}^n a_i \log\frac{\sum_{i=1}^n a_i}{\sum_{i=1}^n b_i}, \tag{5.6}$$

Applied to the above term we get:

$$\begin{aligned}
&\left\|\log\left(\frac{\sum_{y'\in\mathcal{Y}} \exp(\boldsymbol{\theta}_1'\mathbf{f}(\mathbf{x}_v, y', \mathbf{y}_{\setminus v})))}{\sum_{y'\in\mathcal{Y}} \exp(\boldsymbol{\theta}_2'\mathbf{f}(\mathbf{x}_v, y', \mathbf{y}_{\setminus v})))}\right)\right\| \\
\overset{5.6}{\leq} &\left\|\frac{1}{\sum_{y'\in\mathcal{Y}} \exp(\boldsymbol{\theta}_1'\mathbf{f}(\mathbf{x}_v, y', \mathbf{y}_{\setminus v})))} \sum_{y'\in\mathcal{Y}} \exp(\boldsymbol{\theta}_1'\mathbf{f}(\mathbf{x}_v, y', \mathbf{y}_{\setminus v}))) \log\left(\frac{\exp(\boldsymbol{\theta}_1'\mathbf{f}(\mathbf{x}_v, y', \mathbf{y}_{\setminus v})))}{\exp(\boldsymbol{\theta}_2'\mathbf{f}(\mathbf{x}_v, y', \mathbf{y}_{\setminus v})))}\right)\right\| \\
= &\left\|\frac{1}{\sum_{y'\in\mathcal{Y}} \exp(\boldsymbol{\theta}_1'\mathbf{f}(\mathbf{x}_v, y', \mathbf{y}_{\setminus v})))} \sum_{y'\in\mathcal{Y}} \exp(\boldsymbol{\theta}_1'\mathbf{f}(\mathbf{x}_v, y', \mathbf{y}_{\setminus v})))(\boldsymbol{\theta}_1 - \boldsymbol{\theta}_2)'\mathbf{f}(\mathbf{x}_v, y', \mathbf{y}_{\setminus v})))\right\| \\
= &\left\|(\boldsymbol{\theta}_1 - \boldsymbol{\theta}_2) \sum_{y'\in\mathcal{Y}} \underbrace{\frac{\exp(\boldsymbol{\theta}_1'\mathbf{f}(\mathbf{x}_v, y', \mathbf{y}_{\setminus v}))}{\sum_{y''\in\mathcal{Y}} \exp(\boldsymbol{\theta}_1'\mathbf{f}(\mathbf{x}_v, y'', \mathbf{y}_{\setminus v}))}}_{\leq 1 \quad \text{and} \quad \sum_{y'\in\mathcal{Y}} \ldots = 1} \mathbf{f}(\mathbf{x}_v, y', \mathbf{y}_{\setminus v}))\right\| \\
\leq &\left\|(\boldsymbol{\theta}_1 - \boldsymbol{\theta}_2)'\right\| \|\mathbf{f}(\mathbf{x}_v, y'', \mathbf{y}_{\setminus v})))\|_\infty
\end{aligned}$$

Figure 5.2: SGD vs. SAG for constant step size of 0.001



A vertex-wise Lipschitz inequality independent of $v \in V$ is now given by:

$$\begin{aligned}
|g(\boldsymbol{\theta}_1) - g(\boldsymbol{\theta}_2)| &\leq |(\boldsymbol{\theta}_1 - \boldsymbol{\theta}_2)'\mathbf{f}(\mathbf{x}_v, y', \mathbf{y}_{\setminus v})| + \left\|(\boldsymbol{\theta}_1 - \boldsymbol{\theta}_2)'\right\| \|\mathbf{f}(\mathbf{x}_v, y'', \mathbf{y}_{\setminus v}))\|_\infty \\
&\leq 2 \left\|(\boldsymbol{\theta}_1 - \boldsymbol{\theta}_2)'\right\| \|\mathbf{f}(\mathbf{x}_v, y'', \mathbf{y}_{\setminus v}))\|_\infty \\
&\leq \|(\boldsymbol{\theta}_1 - \boldsymbol{\theta}_2)\| \cdot \underbrace{2 \cdot \max_{v \in V} \left\{\|\mathbf{f}(\mathbf{x}_v, y'', \mathbf{y}_{\setminus v}))\|_\infty\right\}}_{:=L}
\end{aligned}$$

It is not surprising to see, that a restriction of the features $\mathbf{f}(\cdot)$ leads to a better behaviour in the numeric optimization since a small Lipschitz constant lets us choose wider step sizes in the SAG optimization procedure. In general the capping of the functions $\|\mathbf{f}\|_\infty < c$ to some constant $c > 0$ can be a convenient solution as discussed in 2.5.2. In this case the stepsize for SAG is bound by:

$$\alpha \leq \frac{1}{2nL} = \frac{1}{4n}$$

For big graphs, Roux et al. [RSB12] show that the step size can be chosen even larger:

$$\alpha \leq \frac{1}{16L} = \frac{1}{32}$$

In our simulations we use $\frac{1}{32} = 0.03125$ as reference step size up to which the basic SAG guarantees convergence.

Problems with basic SAG

The momemtum effect: As can be seen in figures 5.2, 5.3 and 5.4 the Stochastic Average Gradient approach has a better convergence speed in the CRF context then standard SGD as long as the stepsize does not exceed certain bound aries. As seen in figure 5.2 this rate improves when using a large number of past contributions W. Our simulations show that with increased step size the convergence speed increases, as seen in figure 5.3. However larger step sizes cause a behavior that we call the 'momentum effect' and which can be seen observed in figure 5.4. This effect occurs since the basic assumption of the SAG, that the gradient changes gradually and not abruptly is violated. Therefore,

Figure 5.3: SGD vs. SAG Gradient for constant step size of 0.01

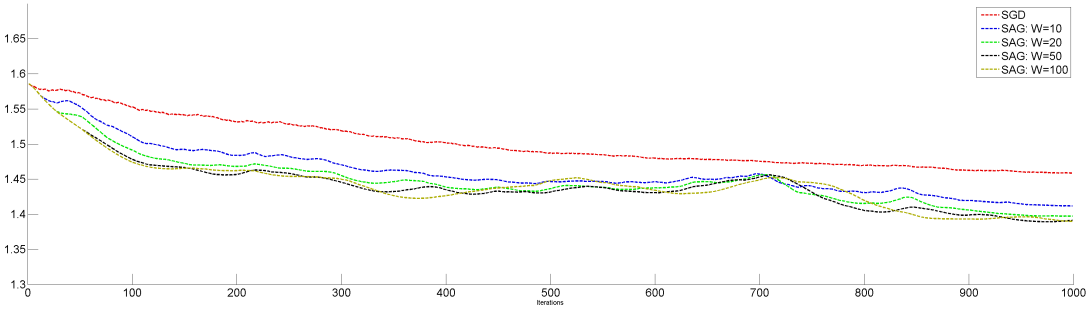
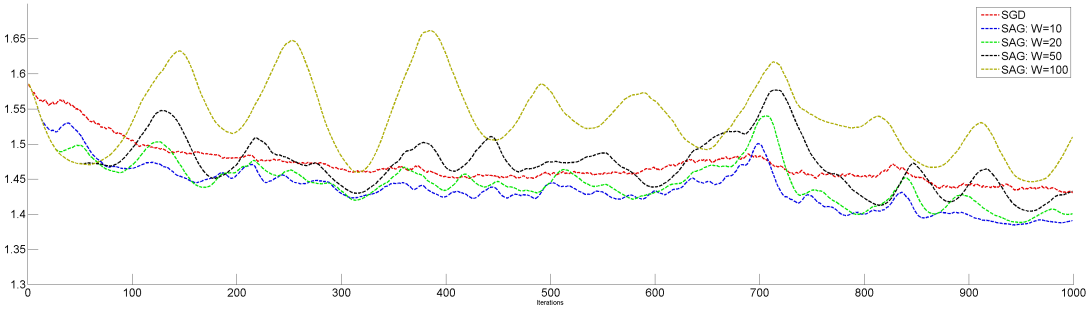
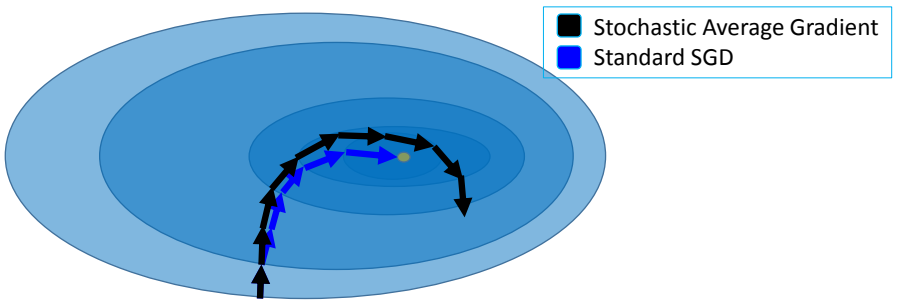


Figure 5.4: SGD vs. SAG for a constant step size of 0.03125



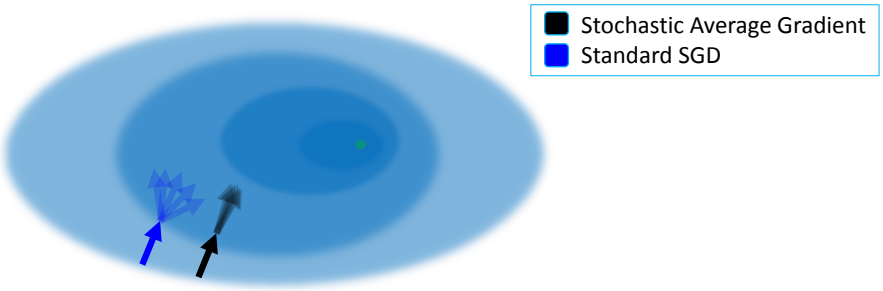
due to the averaging, the change in direction in each iteration is relatively small compared to the optimization step in parameter space. Hence the effect is stronger when averaging over long serious of past contributions (big W).

Figure 5.5: Momentum effect



When aiming for faster convergence and steady convergence behaviour near the optimum, we see four major difficulties when working with basic SAG in the CRF context:
Step sizes: Using small constant step sizes assure the convergence with the constraint

Figure 5.6: SGD vs. SAG, more precise objective function estimation results in lesser stochastic effects in the descend direction.

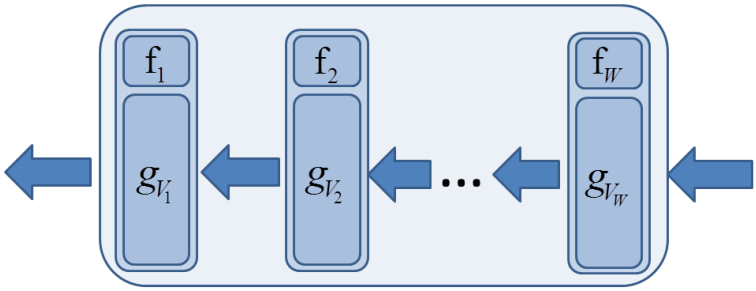


that these step sizes do not exceed certain bounds. This makes it impossible to use more sophisticated step size adaptations. As our simulations (chapter 6) show, the theoretical boundaries for the step size are not completely sharp, but exceeding them by a factor of > 3 often leads to non-converging behaviour of the algorithm due to the momentum effect.
Mini-Batches: Roux et al ([RSB12]) already mention that there are several cases where batch-wise processing is beneficial. Especially if we decide to use larger stepsizes or non-constant stepsizes, the size of the sampled V_t in each iteration plays a key role for it mainly determines the variation of the local approximation of the objective function. Especially when using more sophisticated stepsize adaptions, which might be expensive themselves it is often useful to not use a single vertex for function and gradient updates. Mini-Batches are also of great importance for they usually allow parallel computation of the vertex-contributions.
Basic assumption: The SAG assumes that the gradient does not change abruptly and is therefore capable to use a standard mean of the gradient contributions. This assumption is violated when using bigger stepsizes than the ones guaranteed to converge. This does not allow for local adaptation exceeding these bounds.
Space: Naively storing the gradient contribution for every vertex of the graph would result in a storage complexity of $O(|V| \cdot \dim\boldsymbol{\theta})$, which can easily be too much to store on standard hardware. Using sparse data structures would resolve this problem only to a certain degree, since the complexity of complete runs still remains the same.

Ringbuffer

When applying the SAG method to CRF problems, the asymptotics and the convergence speed found by Roux et al. [RSB12] are still valid. However, as mentioned in 5.4.2 it can be quite costly to store the gradient contributions of all vertices. Since we are looking for solutions for problems where the number of vertices is very high $|V| > 10^7$ and big parameter space $\dim\boldsymbol{\theta} > 10^4$ the storage complexity might easily exceed $|V| \cdot \dim\boldsymbol{\theta} > 10^{11}$. This also makes a complete gradient evaluation very costly.
We propose two changes to cope with this problem:

Figure 5.7: Ringbuffer as used in our SAG modification



- Initialize the $w_v^0 = 0$ since an initial full gradient ist too costly
- Store mini-batch-wise contributions $\sum_{v \in V_s} g_v^i(\boldsymbol{\theta})$
- Discard mini-batch contribution after certain number of iterations W

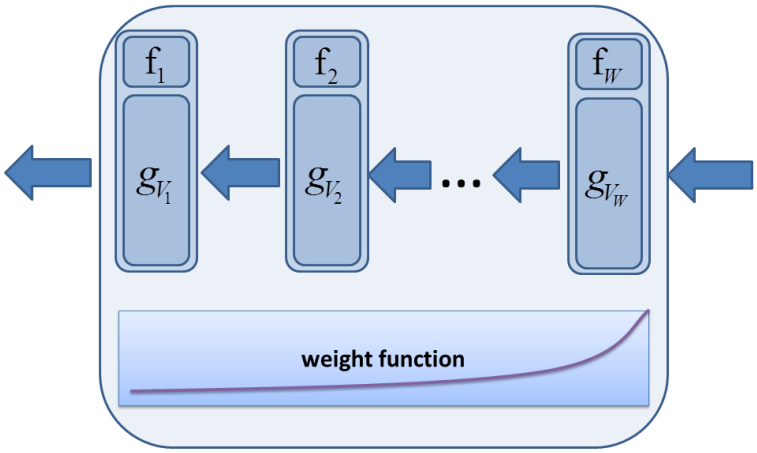
These concepts can be efficiently implemented in a ringbuffer structure as depicted in Figure 5.7. It assumes that information is becoming irrelevant after a certain number of iterations and can hence be discarded. The resulting algorithm is described in algorithm 5.4.2

Algorithm 5.7 Stochastic Average Gradient with mini-batches

Initialize $w_v^0 = 0 \quad \forall v \in V$
while not converged **do**
  Determine a random sub sample $V_s \subset V$
  Empty Ringbuffer R of max size W
  for j = 1 to W **do**
    Update ringbuffer: R.push_back($\sum_{v \in V_s} \nabla g_v(\boldsymbol{\theta})$)
    $G = \frac{1}{|\mathrm{R}|} \sum_{c \in \mathrm{R}} c$
    Determine stepsize α [using V or V_i]
    $\mathbf{y} = \mathbf{x} - \alpha \cdot \frac{G}{\|G\|}$
  end for
end while

The ringbuffer allows storing whole batches of gradient contributions but does not keep track of the vertices that the batch consists of. This eliminates the possibility to actually update a vertex' gradient contribution when the vertex is revisited in a later optimization iteration, for the benefit of a smaller space complexity $O(W \cdot \dim\boldsymbol{\theta})$. Therefore this approach allows for batch-wise processing for the cost of an error that results from having more than one gradient contribution of vertices that lie in more than one batch contribution within the ringbuffer. This effect mainly depends on the size of the ringbuffer. With higher computational effort and higher spatial complexity we could assure that the ringbuffer

Figure 5.8: Weighted ringbuffer



is updated as the standard SAG and does not allow for duplicate vertex contributions. However the next modifications we are introducing are also minimizing this negative effect while enhancing the algorithm itself.

5.4.3 Weighted SAG

The stochastic average gradient algorithm uses the fact that the direction of the gradient usually changes gradually each iteration and not abruptly. It therefore averages over the last W iterations. We propose a slight enhancement to this approach by using a weighted mean over the contribution values instead of just the standard mean. This accounts for the fact that later parameter vectors are more likely to contribute into the direction of the minimum. In addition it weakens the negative effect, that ringbuffers allow more than one contribution per vertex when using batch-wise contributions. The SGD and standard ringbuffer SAG have weighting functions:

- Singular weighting $w(i) = \delta(i = W)$ - Corresponds to SGD
- Constant weighting $w(i) = \frac{1}{W}$ - Corresponds to standard SAG

Other, more practical weighting functions are:

- Polynomial weights $w(i) = i^{\alpha} \frac{1}{\sum_j j^{\alpha}}$
- Exponential weights $w(i) = \alpha^i \frac{1}{\sum_j \alpha^j}$

A good choice of the buffer size W and the weighting function highly depends on the CRF that we are training. In the simulation section 6.2 we show the performance of different weighting functions for different CRF set ups. Overall the results show the

intuitive behaviour i.e. that the convergence is reached later when we apply smoother weighting. On the other hand, using highly exponential weighting leads to situations where the stochastic objective function is very noise and the algorithm cannot determine the minimum within a broad epsilon-band (just like in the SGD case). The weighting allows a problem specific tuning of the ringbuffer:

- $\alpha \gg 1$ - Lesser momentum effect and possibly faster convergence towards a noisy optimum. Therfore convergence only within a broad ϵ-band.
- $1 > \alpha > 0$ - Lesser stochastic effects, higher momentum effect

In general the smoother weighting is beneficial near the optimum, for it reduces the stochastic effects on the objective function. However the momentum effect prevents smoothly weighted SAGs from converging with larger stepsizes. This calls for an additional strategy for adapting the weighting strategy dependent on the status of convergence. We empirically studied ways of using in-sequence variation estimators for the adaptation of the weighting parameters but did not succeed in finding a strategy which can be regarded as problem invariant. In the next section we will therefore present SAG variants that take additional information into account, namely the location in parameter space.

5.4.4 Locally weighted SAG

The positive smoothing effect of showing less jittery results near the point of convergence while it moves slower towards the optimum at the beginning of the process. Choosing an optimal weighting fixed weighting function is usually a highly problem dependent, manual task and also often not possible for a fixed weighting constant α. A transition from one of the above weighting constant to another will introduce more tuning parameters which is also highly dependent on the specific underlying problem. We will therefore concentrate on an adaptive weighting that takes the locations in parameter space into consideration. The weighting function can now defined by a kernel definition in the same way that is known from nonparametric regression or kernel density estimation. For now we want to introduce a parametrized exponential kernel as weighting kernel.

$$d(\boldsymbol{\theta}_t, \boldsymbol{\theta}_i, \alpha) := \exp(-\alpha \|\boldsymbol{\theta}_t - \boldsymbol{\theta}_i\|)$$

where α is a parameter that controls the width of the kernel.

5.4.5 K-Means kernel SAG

Although our simulations show great results for the just introduced locally weighted SAG variant, the approach has one major drawback. Dependency on the euclidean norm is not problem or scale invariant. The weighting is highly dependent on the distance function and therefore also on the scale in parameter space. We investigated different approaches using normalized distances to make the weighting problem and scale invariant. A classic

Figure 5.9: Ringbuffer with local information

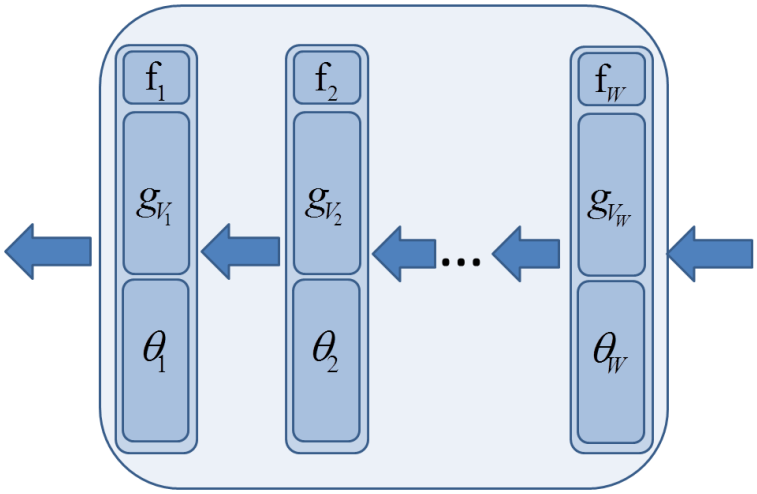
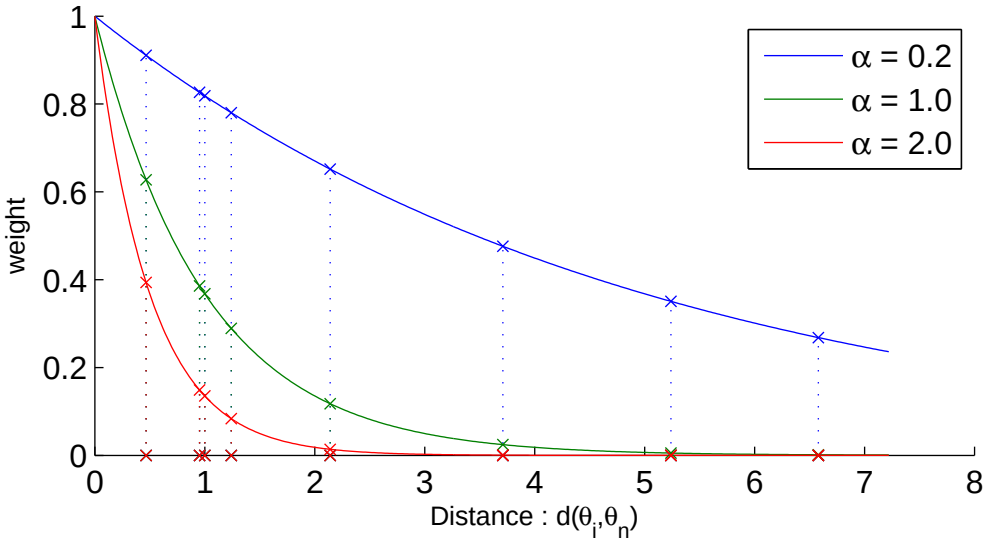


Figure 5.10: Local exponential weighting kernel



robust approach for that is to use a k-means kernel. Let θ_k denote the $k-$th nearest neighbour of the current parameter vector θ_n in iteration n. The k-means kernel is defined by the weighting function:

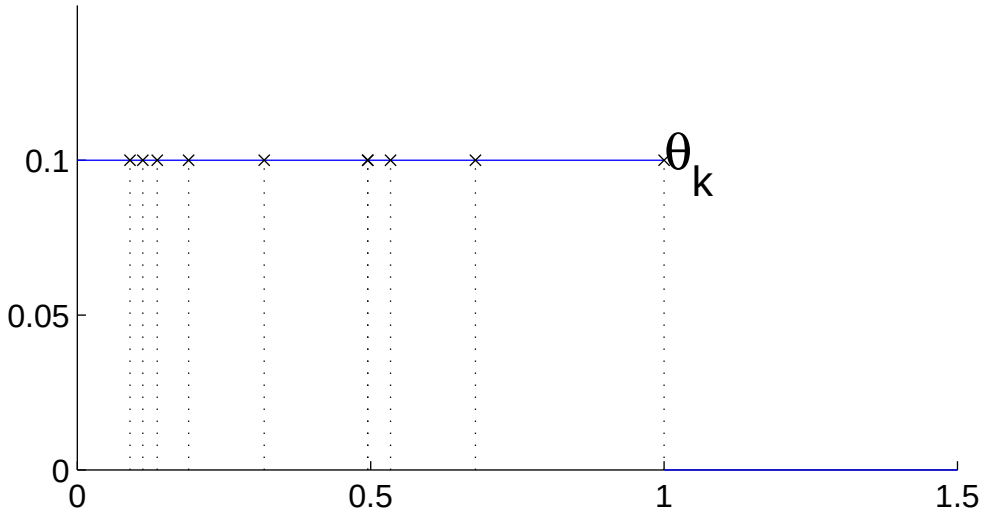
$$w(\boldsymbol{\theta}_i) := \delta(d(\boldsymbol{\theta}_i, \boldsymbol{\theta}_n) < d(\boldsymbol{\theta}_k, \boldsymbol{\theta}_n))$$

The weighting curve is depicted in figure 5.11. The parameter k, denoting the range of neighbors that are taken into consideration by the kernel is a scale invariant tuning parameter. One of the strengths of SAG is to use past gradient contributions to reduce the jitter of the objective function near the optimum. In the ringbuffer SAG this quality depends on the buffer size W. Using a k-means kernel reduces the influencing past contributions to k and therefore limits this positive effect of the SAG. We can define a small overlap amount $\gamma > 0$ to achieve a consideration of more than k contributions near the optimum by simply using a modified weight function in the form:

$$w(\boldsymbol{\theta}_i) := \delta(d(\boldsymbol{\theta}_i, \boldsymbol{\theta}_n) < d(\boldsymbol{\theta}_k, \boldsymbol{\theta}_n) * (1.0 + \gamma))$$

This again introduces another tuning parameter, which is at least scale invariant. Since the effect of γ strongly depends on the stochastic behavior of the objective function it is not to be analytically chosen in general. Values between 0.2 and 0.9 proof to be effective in most observed cases.

Figure 5.11: k-means weighting



5.4.6 Robust locally weighted SAG

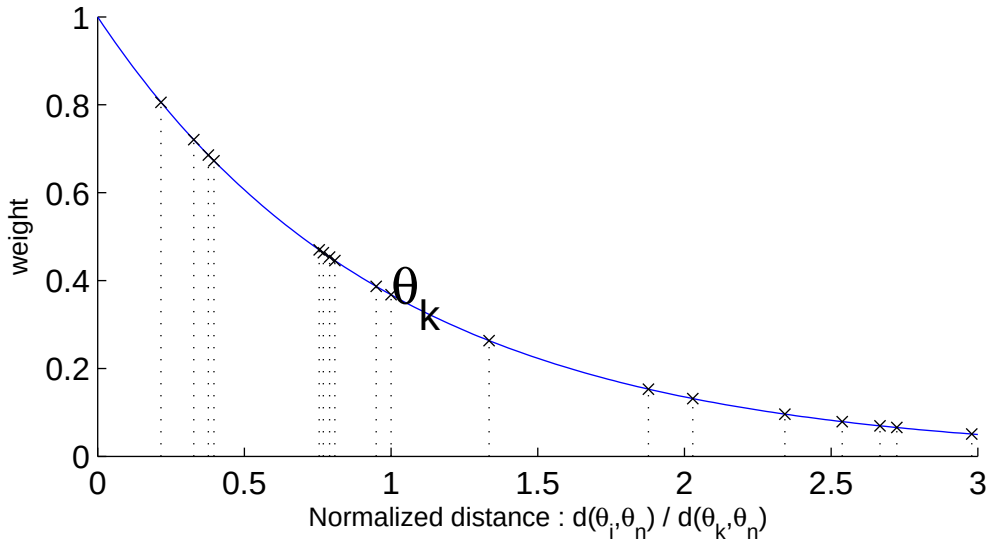
Using the k-means kernel is scale independent, however our simulations show that it is not suited for big step sizes. We can define a robust locally weighted SAG variant by combin-

ing the last two proposed SAG enhancements 5.4.5, 5.4.4. The exponential weighting is normalized by the distance to the k-th nearest neighbor of the current parameter $\boldsymbol{\theta}_n$.

$$w(\boldsymbol{\theta}_i) := \exp\left(-\frac{d(\boldsymbol{\theta}_i, \boldsymbol{\theta}_n)}{d(\boldsymbol{\theta}_k, \boldsymbol{\theta}_n)}\right)$$

This results in a weighting function as depicted in 5.4.6. One immediate benefit is, that the overlap parameter γ from the k-means approach as well as the weighting parameter α from the exponential kernel can be used with this approach as well, but they can be interpreted completely problem and scale independent. In our simulation chapter 6 we also demonstrate that this approach is robust against choosing high step sizes. The choice of k is not critical and we omit this degree of freedom in most simulations since in broad ranges $2 < k < 20$ the convergence behavior is very similar.

Figure 5.12: Local robust kernel - depicted with a set of examplary distances



5.5 Convergence measures

The above procedures all numerically approach the optimum of an objective function. As convergence measures there are several possibilities.

5.5.1 Parameter space metric

A very straight forward convergence measure is to use a norm on the change in model parameter. Very popular examples are

$$\|\boldsymbol{\theta}\| = \sum |(\theta_i)| \quad \text{or} \quad \|\boldsymbol{\theta}\| = \sum \theta_i^2 \quad \text{or} \quad \|\boldsymbol{\theta}\| = \max_i \theta_i$$

This implies that the model parameter $\boldsymbol{\theta}$ is of equal importance in all dimensions. This can be achieved by using normalized features (2.5.2). In many applications the features are defined by indicator functions and thus the model parameter is equivalent in all dimensions. Convergence is then defined by the metric on the parameter difference and a threshold ϵ.

$$\|\boldsymbol{\theta}_i - \boldsymbol{\theta}_{i-1}\| < \epsilon$$

With stochastic optimization it is best practice to iterate until the threshold has been undergone a fixed number of times T. This prevents accidental algorithm termination due to a noise objective function. A convergence criterion for a stochastic optimization procedure can therefore be defined by

$$\max_{0<t\leq T} \|\boldsymbol{\theta}_{i-t} - \boldsymbol{\theta}_{i-t-1}\| < \epsilon \tag{5.7}$$

5.5.2 Likelihood ratio

Whenever there is little information about the observation vector $\mathbf{X}$ normalization of the features can be difficult. This applies especially to unbound continuous observation data. A normalization by taking information from the available data always leads to a systematic overfitting. In such cases the model parameter θ can be highly homogeneous and the convergence by looking at parameter metric is invalid. To keep it as general as possible we assume we want to compare two parameter sets $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_2$ by using the likelihood ratio. The CRFs likelihood function for a given sample set $(\mathbf{x}, \mathbf{y})$ is given by:

$$\begin{aligned} L(\boldsymbol{\theta}) = p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{y}) &= \frac{1}{Z(\mathbf{x})} \prod_v \exp\left(\boldsymbol{\theta}' \mathbf{f}(y_v, \mathbf{y}_{N(v)}, \mathbf{x}_v)\right) \\ &= \frac{1}{Z(\mathbf{x})} \exp\left(\boldsymbol{\theta}' \left(\sum_v \mathbf{f}(y_v, \mathbf{y}_{N(v)}, \mathbf{x}_v)\right)\right) \end{aligned}$$

The likelihood ratio is:

$$\begin{aligned} \Lambda := \frac{L(\boldsymbol{\theta}_1)}{L(\boldsymbol{\theta}_2)} &= \frac{Z_{\boldsymbol{\theta}_2}(\mathbf{x})}{Z_{\boldsymbol{\theta}_1}(\mathbf{x})} \exp\left((\boldsymbol{\theta}_1 - \boldsymbol{\theta}_2)' \left(\sum_v \mathbf{f}(y_v, \mathbf{y}_{N(v)}, \mathbf{x}_v)\right)\right) \\ &= \exp\left((\boldsymbol{\theta}_1 - \boldsymbol{\theta}_2)' \left(\sum_v \mathbf{f}(y_v, \mathbf{y}_{N(v)}, \mathbf{x}_v)\right)\right) \\ &\quad \cdot \mathbf{E}\left[\exp\left((\boldsymbol{\theta}_1 - \boldsymbol{\theta}_2)' \left(\sum_v \mathbf{f}(y_v, \mathbf{y}_{N(v)}, \mathbf{x}_v)\right)\right)\right]_{\boldsymbol{\theta}_2} \end{aligned}$$

This is usually not tractable, and we face the same numerical difficulties that arise when using classic maximum likelihood training 4.1. Therefore we propose to use the pseudo-likelihood ratio defined by

$$\begin{aligned}\Lambda_{\mathcal{P}} := \frac{L_{\mathcal{P}}(\boldsymbol{\theta}_1)}{L_{\mathcal{P}}(\boldsymbol{\theta}_2)} &= \prod_{v\in V} \frac{\exp\left(\boldsymbol{\theta}_1'\mathbf{f}(y_v, \mathbf{y}_{\setminus v}, \mathbf{x}_v)\right)}{\sum_{y'\in\mathcal{Y}} \exp\left(\boldsymbol{\theta}_1'\mathbf{f}(y', \mathbf{y}_{\setminus v}, \mathbf{x}_v)\right)} \frac{\sum_{y'\in\mathcal{Y}} \exp\left(\boldsymbol{\theta}_2'\mathbf{f}(y', \mathbf{y}_{\setminus v}, \mathbf{x}_v)\right)}{\exp\left(\boldsymbol{\theta}_2'\mathbf{f}(y_v, \mathbf{y}_{\setminus v}, \mathbf{x}_v)\right)} \\ &= \prod_{v\in V} \exp\left((\boldsymbol{\theta}_1 - \boldsymbol{\theta}_2)'\mathbf{f}(y_v, \mathbf{y}_{\setminus v}, \mathbf{x}_v)\right) \frac{\sum_{y'\in\mathcal{Y}} \exp\left(\boldsymbol{\theta}_2'\mathbf{f}(y', \mathbf{y}_{\setminus v}, \mathbf{x}_v)\right)}{\sum_{y'\in\mathcal{Y}} \exp\left(\boldsymbol{\theta}_1'\mathbf{f}(y', \mathbf{y}_{\setminus v}, \mathbf{x}_v)\right)}\end{aligned}$$

The same effect as in the pseudo-likelihood based training occurs. The summation of realizations of $y' \in \mathcal{Y}$ is performed locally at each vertex v and is therefore tractable. Instead of using the pseudo-likelihood ratio it is also possible to use the log pseudolikelihood. However depending on the sample $(\mathbf{x}, \mathbf{y})$ and the model parameter $\boldsymbol{\theta}$ this might lead to numeric instabilities, since values of $\log L_{\mathcal{P}}(\boldsymbol{\theta})$ are usually closer to zero. Therefore we prefer the use of the pseudo-likelihood ratio over the logarithmic variant.
The above definition assumes that the pseudolikelihood function is evaluated on the same set of vertices V for both model parameters. In stochastic algorithms this is usually not the case. A more general ratio definition is therefore:

$$\Lambda_{\mathcal{P}}(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, V_1, V_2) := \prod_{v\in V_1} \left(\frac{\exp\left(\boldsymbol{\theta}_1'\mathbf{f}(y_v, \mathbf{y}_{\setminus v}, \mathbf{x}_v)\right)}{\sum_{y'\in\mathcal{Y}} \exp\left(\boldsymbol{\theta}_1'\mathbf{f}(y', \mathbf{y}_{\setminus v}, \mathbf{x}_v)\right)} \right) \prod_{v\in V_2} \left(\frac{\sum_{y'\in\mathcal{Y}} \exp\left(\boldsymbol{\theta}_2'\mathbf{f}(y', \mathbf{y}_{\setminus v}, \mathbf{x}_v)\right)}{\exp\left(\boldsymbol{\theta}_2'\mathbf{f}(y_v, \mathbf{y}_{\setminus v}, \mathbf{x}_v)\right)} \right)$$

A straight forward convergence criterion is defined analogue to the parameter space convergence criterion for stochastic algorithms, by a threshold $\epsilon > 0$ and a fixed number $T > 0$:

$$r_1(i) := \max_{0<t\leq T} \|\Lambda_{\mathcal{P}}(\boldsymbol{\theta}_{i-t-1}, \boldsymbol{\theta}_{i-t}, V_{i-t-1}, V_{i-t})\| < \epsilon \tag{5.8}$$

Using the same set of vertices $V_1 = V_2$ defines a much stronger convergence criterion since it also requires consecutive model parameters to achieve a good likelihood with different sets of vertices. This stronger variant is defined by:

$$r_2(i) := \max_{0<t\leq T} \|\Lambda_{\mathcal{P}}(\boldsymbol{\theta}_{i-t-1}, \boldsymbol{\theta}_{i-t}, V_{i-t-1}, V_{i-t-1})\| < \epsilon \tag{5.9}$$

Monitoring the difference of $r_1(i)$ and $r_2(i)$ can be used to reveal poorly chosen vertex sets in stochastic algorithms.

5.6 Summary

In this chapter we introduced the most common nonlinear optimization techniques, stochastic approaches and specialized in the SAG method. The stochastic optimization methods are necessary whenever the objective function is hard to calculate, which can easily be the case when dealing with graphical models on huge graphs. Within the field of stochastic approaches and the introduced methods we can summarize:

- SAG is proven to converge fast (exponential) for small stepsizes
- Using bigger stepsizes leads to a 'momentum effect' and slow initial convergence
- Using bigger mini-batches defer the momentum effect
- Weighted averaging can cope with slow convergence (and also the momentum effect)
- Optimal weighting can only be achieved by reacting to the current state of convergence
- We use metrics in parameter space as parameter for weighting
- This is highly dependent on the parameter space and also has a problem specific tuning parameter
- We normalize the distance by k-means, and use weighting, which allows the use of bigger stepsizes
- Using stochastic optimization requires for specific convergence measures. In the following simulation studies we deal with normalized features and can use metrics in parameter space

6 Evaluation

In this chapter we study the strengths and weaknesses of the numeric optimization techniques and especially the newly introduced SAG variants (5.4.3,5.4.4,5.4.6). Our goal is to evaluate CRFs for practical applications in road quality modeling. We design different sets of graphs and artificial variable set ups, that allow us to benchmark the optimization techniques in relevant scenarios. We focus solely on the training of the model and cover the classification and prediction qualities in chapter 7.

6.1 Simulation setup

To systematically benchmark the numeric techniques from the optimization chapter 5 we define a series of set ups that define the algorithms, the objective functions and the underlying CRFs. First we split the CRF's definition into three characteristics:

- Underlying graph topology
- Target variable and observable variables
- Feature definition

6.1.1 Topology setup

Street graph definition

We are using optimization techniques for general CRFs and therefore do not assume any constraints on the the underlying graph's topology. When creating a set of artificial graphs for CRF evaluation there are many degrees of freedom that can be taken into consideration. The mathematical definition of a graph is the tuple of its vertices and edges $G = (V, E)$. The most prominent degree of freedom is the amount of vertices $n = |V|$. Since there is no topology induced by the shear number of vertices all other degrees of freedom are ways to define edges between the vertices. Let the cardinality $C(v)$ of a vertex denote the number of adjacent vertices or equivalently the number of half edges originating at the vertex,

$$C(v) := \left|\{e = (w_1, w_2) \in E \mid w_1 = v \quad \text{or} \quad w_2 = v\}\right| \quad \text{with} \quad v \in V.$$

The set of vertices, without any topology or local information, are merely an index number and completely interchangeable. Therefore a graph's topology can be described to a large

degree by the series of cardinalities of the vectors. In order to keep the complete set of possible graph setups to a limited amount, we will make one further assumption. We assume that the distribution of cardinalities are of a spatially random nature and the topology can be represented by a frequency statistics of the cardinalities.

Vertex cardinalities in VMC

To obtain a set of realistic artificial topologies we make use of real world road network topologies. We use the dataset of the VMC project (see section 1.3) which is based on OpenStreetMap (http://www.openstreetmap.org) road information. An important meta information of a road in this dataset is the road's functional class. OpenStreetMap defines a variety of different road classes. VMC aggregates those classes to define a set of four distinct classes as indicated in Table 6.1. In the VMC context this categorization is used to

Table 6.1: VMC and OSM road classification

VMC class	OpenStreepMap class
motorway	motorway
major	trunk, primary
intermediate	secondary,tertiary
minor	unclassified,residential,service

define different measures of curvature or hilliness or to assume different driving behavior for each class. Here we are using the road categorization to analyze the vertex cardinality distribution within each class. The resulting cardinality distributions of the joint dataset of Germany, Finland and Sweden can be seen in Figures 6.1 and 6.2.

Figure 6.1: Adjacent edges per vertex on the road-network-graph for Germany,Finland and Sweden

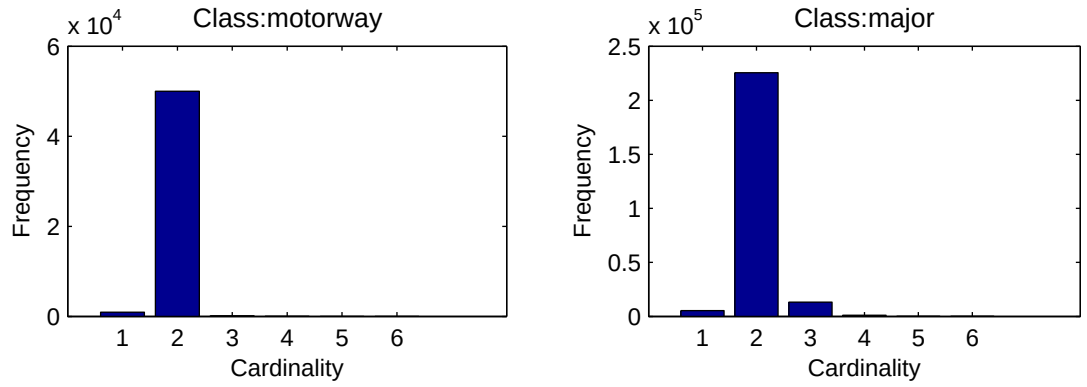
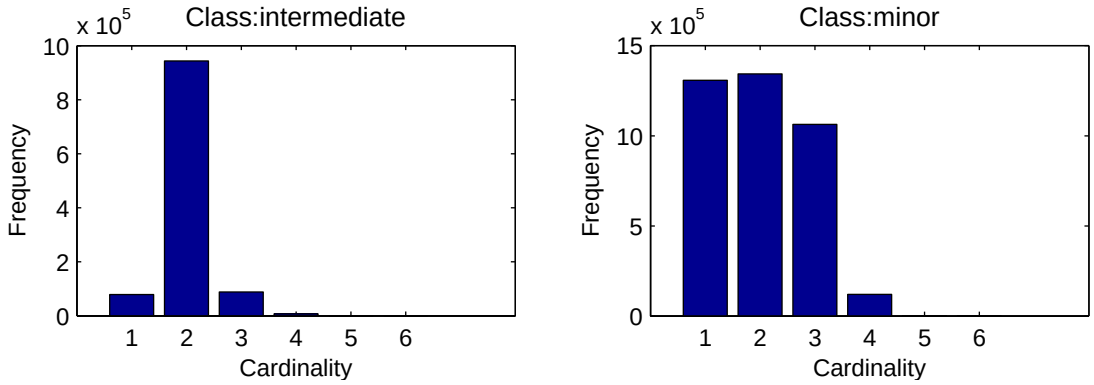


Figure 6.2: Adjacent edges per vertex on the road-network-graph for Germany,Finland and Sweden (lower functional classes)



The histograms in 6.2 and 6.2 show that the classes 'major' and 'intermediate' are of very similar cardinality distribution. We define three categories of histograms which we use as prototypes for the artificial topologies:

1. Street networks with negligible inter road connectivity
   As seen in the histogram of the motorways (Figure 6.1), there are street types where inter road connectivity is not present or statistically not significant
   (no cardinalities > 2).

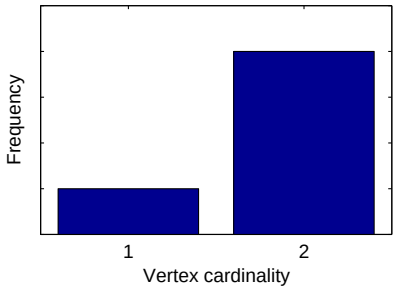
2. Streets with low connectivity
   Major roads and intermediate roads show a small, but not negligible amount of interconnectivity (cardinalities > 2)

3. Street network with high connectivity
   Minor roads, like inner city streets, show a high amount of connections between roads.

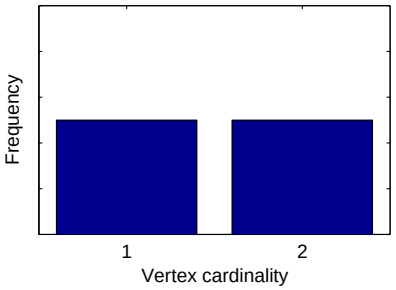
We bear in mind that the data behind the histograms (6.1 and 6.2) is preprocessed OpenStreetMap data used by VMC. This preprocessing step involves a road segmementation which accumulates atomic road parts to bigger objects. Such a road object in the database is defined by a set of constraints which is defined by VMC. Assuming different constraints (which is valid) will lead to a different segmentation. In the section 7.2.1 we will discuss criteria for road segmentation. A more coarse segmentation leads to an absolute decrease of nodes of cardinality two, since those vertices are omitted when smaller end-to-end road segments are joined. By thinning out the set of vertices with cardinality two we can define three additional relevant histograms, which resemble the same road network scenarios as above but assuming different segmentation constraints.

Table 6.2: Prototypical cardinality distributions for the artificial road network set ups

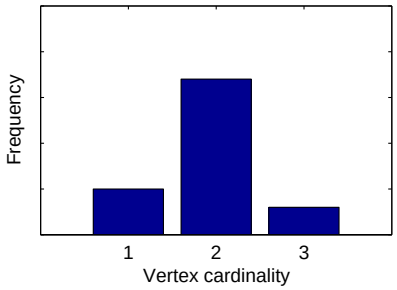
T1 - Motorway



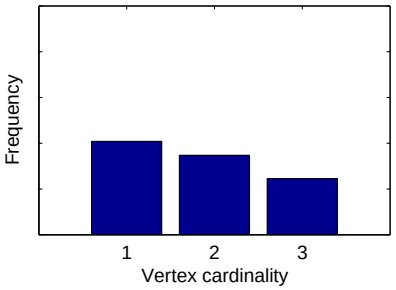
T4 - Motorway (coarse)



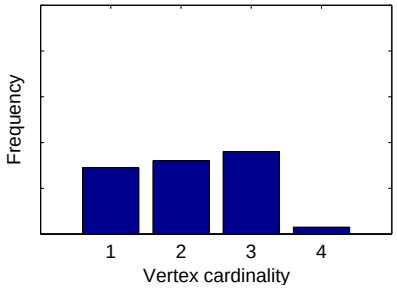
T2 - Major Roads



T5 - Major Roads (coarse)



T3 - Minor Roads



T6 - Minor Roads (coarse)

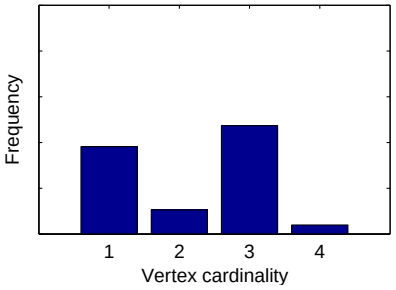
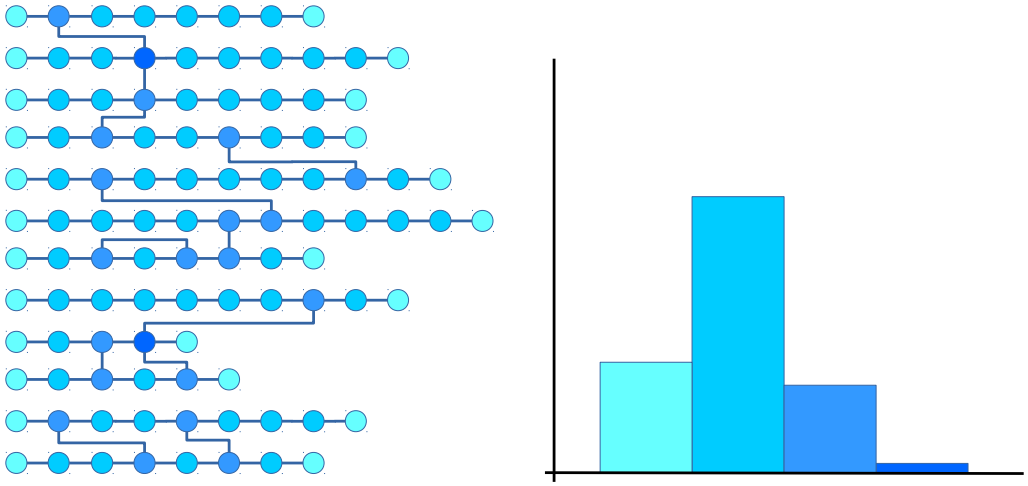


Figure 6.3: Schematic of the street based topology setup (according to histogram)



Setup algorithm

Given a histogram of vertex cardinalities as in the figures 6.1 and 6.2 we specify the following graph setup. Let $h(i)$ denote the amount of vertices of cardinality i.

- The street count is determined by the amount of loose ends $n := h(1)/2$
- The remaining nodes are inner nodes and therefore determine the average length of the streets $m := mean(L) = 2 + \frac{1}{n}\sum_{i=2}^{\infty} h(i)$
- The distribution of nodes per street with the mean m is highly dependent on the location and on the road segmentation paradigms. To simplify we assume a Gaussian distribution $N(m, \sigma^2)$ with a given variance σ^2. High values of σ correspond to areas of varying structure while small values of σ describe very regular street structures as we find it in American city centers.
- We create n artificial roads of length according to $N(m, \sigma^2)$. The histogram values $h(i)$ for $i > 2$ are interpreted as connections between roads and are inserted randomly between the roads.

This way we receive a road network in the form of the schematic depicted in Figure 6.3 This way we defined a way to set up real life based artificial graphs by reducing completely arbitrary graphs to the set defined by three degrees of freedom.

- Size of the graph, defined by the number of vertices [n]
- Functional class, defined by cardinality histogram [T1-T6]
- Uniformity of street lengths, defined by a variance figure [σ]

6.1.2 Variable setup

In the context of classification on graphs each vertex is assigned a tuple of one scalar target variable and a vector of observable information. In this section we define five sets of artificially and randomly chosen variables. The goal is to research the influence of the problem complexity and the convergence speed. Further will define a set of variable that will later be used in the evaluation of the different SAG variants. The degrees of freedom that are of interest are:

- Range of the discrete observation variables
- Range of the target variable
- Discrete / continuous observation

The setups are summarized in Table 6.3. The labels in the last row of Table 6.3 give an intuition what such variables are likely to resemble.
The first setup V1 is a small set of three discrete observations with small ranges. However the feature function set of all possible indicator functions on combinations has already > 1000 features. The second setup V2 is used to analyze the effect of extended ranges of observations. Therefore the ranges of the observations are each larger by a factor of five. Usually we will incorporate continuous data in application. The third to fifth setup V3 treats the third observation as normalized continuous quantity. These last three setups also include a variation of the label range. In our application we will mostly stick to a range of five, but as we will see in section 7.2.2 it can be useful to change this range.
In all setups the variables are generated by independent uniform sampling. The independence is not necessary since CRFs can cope with dependencies very well.

Table 6.3: Set ups for the variables per vertex of the graph

Setup	$Y \in$	X_0	X_1	X_2	$\dim(\boldsymbol{\theta})$
V1	$\{0,\dots,4\}$	$\in \{1,\dots,4\}$	$\in \{1,\dots,3\}$	$\in \{1,\dots,10\}$	1102
V2	$\{0,\dots,4\}$	$\in \{1,\dots,20\}$	$\in \{1,\dots,15\}$	$\in \{1,\dots,50\}$	85682
V3	$\{0,\dots,4\}$	$\in \{1,\dots,4\}$	$\in \{1,\dots,3\}$	$\sim U(0,1)$	280
V4	$\{0,1\}$	$\in \{1,\dots,4\}$	$\in \{1,\dots,3\}$	$\sim U(0,1)$	80
V5	$\{0,\dots,14\}$	$\in \{1,\dots,4\}$	$\in \{1,\dots,3\}$	$\sim U(0,1)$	587
	IRI	Pavement type	Number of lanes	Traffic	

6.1.3 Feature setup

Functions

Since we are using feature functions of the form defined in 2.4.2 we can define the functions to be either vertex-based or edge-based. In the variable set ups V1-V2 (6.1.2) the observation vector is discrete and the local feature functions can be defined as

$$F_{local} := \{f_{y',\mathbf{x}'} | y' \in \mathcal{Y}, \mathbf{x}' \in \mathcal{X}\} \quad \text{with}$$
$$f_{y',\mathbf{x}'}(y_v, \mathbf{x}_v) = \mathbb{1}(y_v = y' \wedge \mathbf{x}_v = \mathbf{x}') \quad \text{with} \quad y' \in \mathcal{Y} \quad \text{and} \quad \mathbf{x}' \in \mathcal{X}.$$

In the set ups V3-V5 we use a linear monome approximation to the functional dependency of the continuous valued X_2. This is described in 2.4.2.

$$F_{mon} := \{f_{y',x'_0,x'_1,n} | y' \in \mathcal{Y}, (x'_0, x'_1, \cdot) \in \mathcal{X}, n \in \{0,1\}\} \quad \text{with}$$
$$f_{y',x'_0,x'_1,n}(y_v, \mathbf{x}_v) = \mathbb{1}(y_v = y' \wedge x_{v,0} = x'_0 \wedge x_{v,1} = x'_1) \cdot x_2^n.$$

For the edge-wise potentials we choose two edge-wise feature functions:

$$f_{dist} = d(y_v, y_w) = \exp(|y_v - y_w|) \tag{6.1}$$
$$f_{\delta} = \mathbb{1}(y_v = y_w) \tag{6.2}$$

where f_{dist} usually is assigned a high negative weight and f_{δ} a high positive weight. The total amount of feature functions for each variable setup can be found in Table 6.3.

Feature weights

In section 6.1.2 we defined the bounds for the target variable 'IRI'. In our simulations the realization values are determined using MCMC as described in (3.3) using the features defined in this section. This requires further to define a set of 'true' weights for the feature functions. Since we do not want assume any knowledge about the nature of the connection between the observations and the label, we sample these weights randomly. We define a set of random distributions used for the weights.

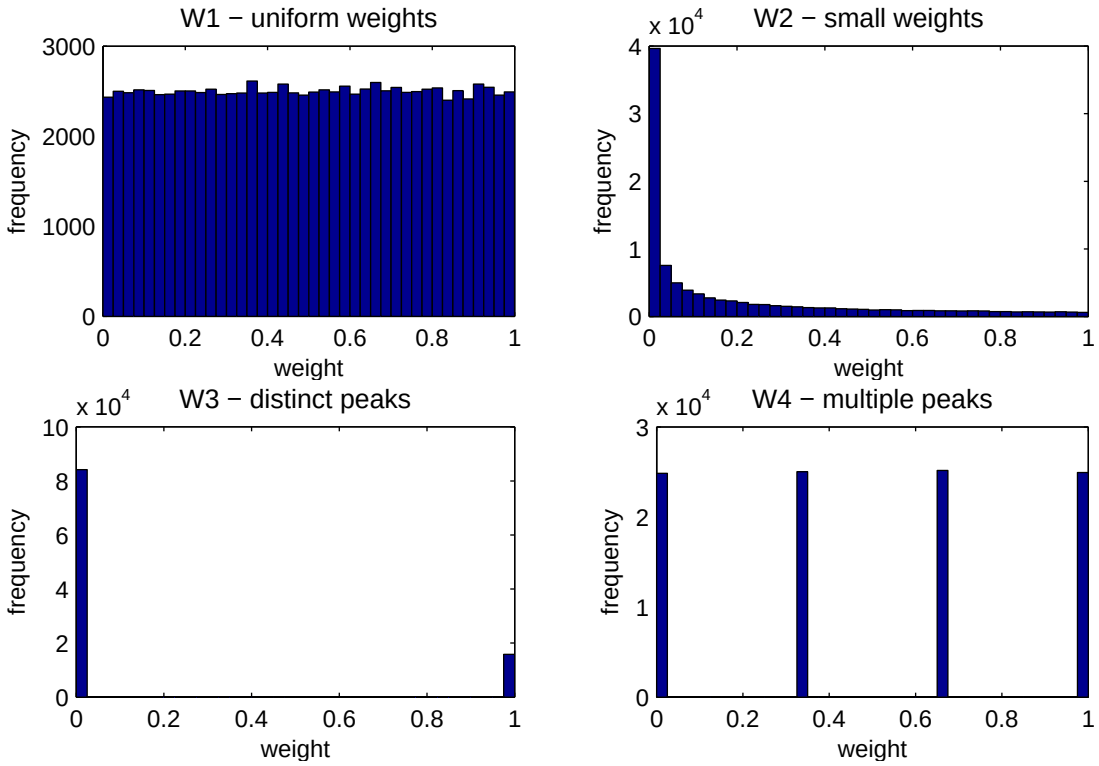
W1 Uniformly distributed weights
$w_i \sim U[0,1]$

W2 Transformed weights for high probability of small values :
$w_i = X^4 \quad \text{with} \quad X \sim U[0,1]$

W3 Distribution with peaks at zero and 1, realized by taking the rounded values of W1
$w_i = [X^4] \quad \text{with} \quad X \sim U[0,1]$
where $[\cdot]$ denotes rounding to the nearest integer number

W4 Distribution with several (we chose 4) peaks:
$w = [X - 0.5]/3 \quad \text{with} \quad X \sim U[0,4]$

Figure 6.4: Histograms for randomly chosen weights according to the setups W1-W4



The corresponding histograms for randomly chosen weights according to our setups are depicted in Figure 6.4. We make one further adjustment by assuming that the topology of the graph is of high importance. This is achieved by setting a high weight on edge-based features. For our simulations we therefore always set:

$$w_{dist} = -1 \tag{6.3}$$

$$w_{\delta} = 1 \tag{6.4}$$

where w_{dist} denotes the weight for the edge-wise target variable distance feature f_{dist} and w_{δ} analogously for f_{δ}.

6.1.4 Objective functions

In chapter 4 we introduced several different approaches to CRF training. For CRFs on big graphs essentially pseudo-likelihood is the method of choice. Additionally contrastive divergence also results in an objective function in the form of a sum over all vertices. Therefore the optimization techniques can be applied in the exact same manner as for pseudo-likelihood functions and we stick to the pseudo-likelihood as objective function. One degree of freedom regarding the objective function is the penalty term that is added in order to regularize the model. We introduced L1,L2 and elastic net regularization in chapter 5. We use the following abbreviations in the setups:

O0 Plain Pseudoliklihood

O1 Pseudolikelihood + $\lambda\cdot$ L1 penalty term

O2 Pseudolikelihood + $\lambda\cdot$ L2 penalty term

O3 Pseudolikelihood + $\lambda\cdot$ (0.5$\cdot$ L1 + 0.5$\cdot$ L2)

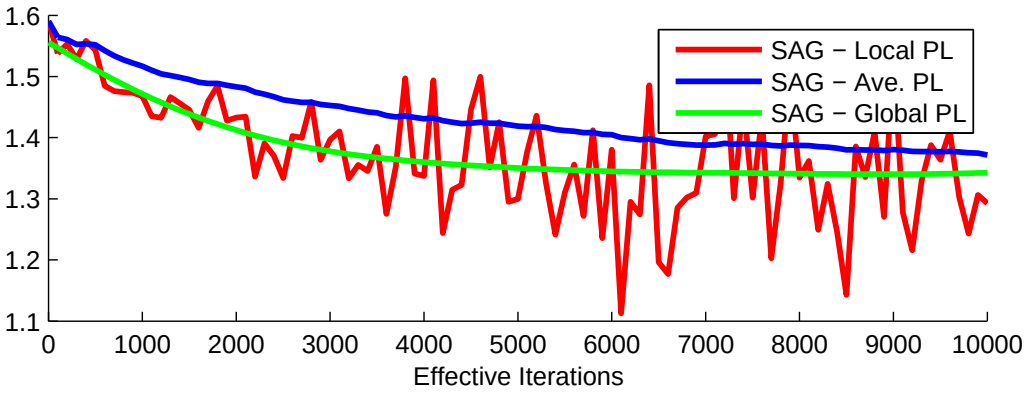
O4 Pseudolikelihood + $\lambda\cdot$ (0.1$\cdot$ L1 + 0.9$\cdot$ L2)

O5 Pseudolikelihood + $\lambda\cdot$ (0.9$\cdot$ L1 + 0.1$\cdot$ L2)

6.1.5 Interpreting simulations

In the simulation studies we compare different approaches with one another. The objective function that is supposed to be minimized is a sum of a negative log-pseudolikelihood term and a regularization term. The likelihood term in stochastic approaches again is a sum of likelihood contributions over a set of vertices of interest. If we analyse the objective function that is minimized in a specific iteration of the optimization procedure we observe different scales of function values dependent on the choice of regularization term and optimization procedure (deterministic or stochastic). In order to make different approaches, with different batch sizes and ringbuffers comparable we define the following target values:

Figure 6.5: Global, averaged and local objective Functions of a sample SAG optimization



- **Global Value:** The value of the negative log-pseudolikelihood function divided by the number of vertices that have been used
- **Averaged value:** For SAG approaches: Denotes the value that results from evaluating the ringbuffer based weighting according to the specified weighting function, divided by the number of vertices. This value is the SAG approximation for the above defined 'Global Value'.
- **Local Value:** The last value in the ringbuffer entry. It can be used to visualize the uncertainty of the objective function locally evaluated. Further it can be used to compare SGD and SAG variants and it can explain some phenomena of the SAG curves.

We keep in mind that the global value is usually computationally expensive to evaluate on big graphs. The stochastic approaches do not use the global value for optimization but the averaged / local values of the objective function. In our simulation runs we invested the high computational effort to always calculate the global values as well. Therefore the results always depict the values of the global objective, while the values used for the optimization are the averaged / local approximations. By dividing by the number of vertices we achieve a comparability of likelihood functions for the complete set of vertices and for evaluations on subsets. In the simulation results we find the global objective values per vertex on the y-axis of the graphs.

In order to make our results comparable in terms of efficiency or computation time, we can make use of different quantities:

- **Iteration:** The optimization iteration, disregarding the amount of linesearch steps or the number of feature functions actually evaluated (disregarding thresholding) The number of iterations can be useful when comparing different mini-batch sizes in stochastic approaches and assuming that parallel computing of all mini-batch elements is possible regardless of its size.

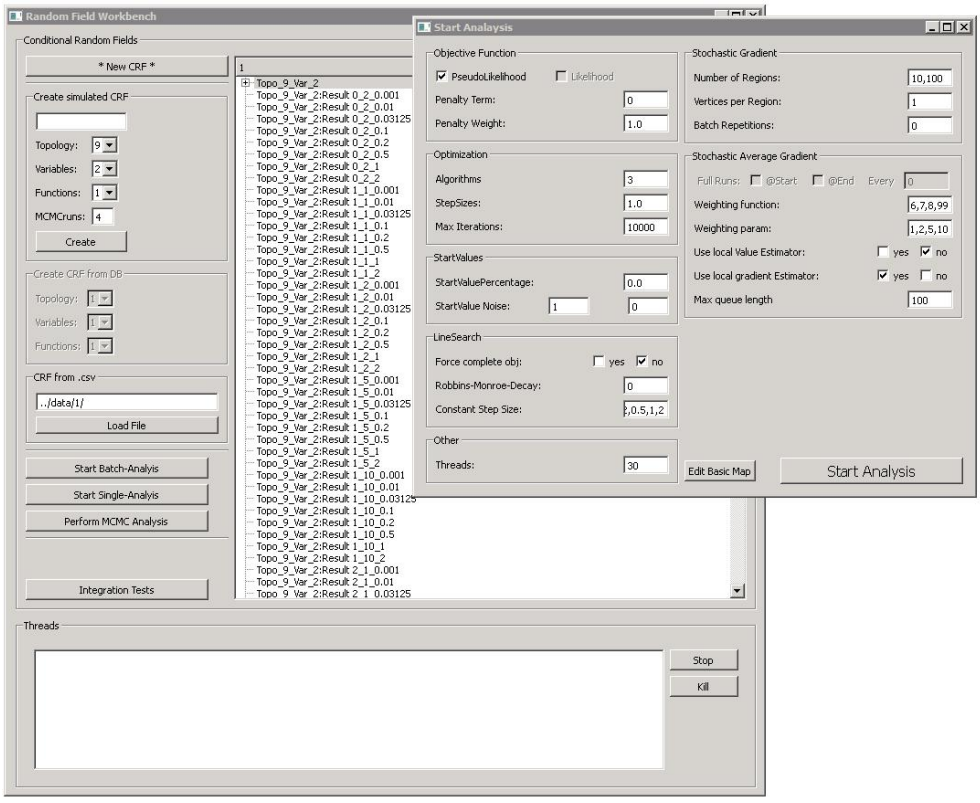
- **Effective Iteration:** The number of iterations multiplied by the number of vertices that were considered. This accounts for the fact that stochastic optimization procedures' iterations are of lower computational complexity.
- **Function evaluations:** The number of feature functions that have been evaluated. This number is reduced if functions are deactivated using thresholding. The function evaluations do incorporate parallel programming benefits for mini-batch approaches.
- **Time:** The execution time until convergence is a valid descriptor for the efficiency of a selected algorithm together with the specific implementation and the system that it is executed. However the high dependency on the latter factors makes it not favorable for most analysis.

These quantities are usually used as categories for the x-axis of the result graphs for our simulations.

6.2 Simulations

The set ups that we introduced in 6.1 contain a large number of degrees of freedom. In the first two simulation studies we examine which set up parameters are essential for the optimization performance. This narrows down the set ups to an essential set. This remaining set is used in the following studies to analyze the optimization methods introduced in chapter 5. Every simulation is described in words and in a table describing the setup and optimization parameters used. In the table we use set notation for degrees of freedoms (using curly brackets). The full results for all simulation runs can be found in the appendix.
In order for the simulations to work, we implemented a simulation environment with C++, with multi-threading capabilities. The hardware that was used for the studies is a 40-core high performance server. This allows us to run simulations on complex random fields and still evaluating the global likelihood function, which would not be efficiently computable on standard desktop hardware as targeted in VMC. This is done to show the actual optimization process instead of only local approximations, which were used as described in 6.1.5. A screenshot of the simulation environment is shown in Figure 6.6.

Figure 6.6: Simulation Environment



6.2.1 Influence of the artificial topology

We introduced a way to set up real life based artificial graphs by reducing completely arbitrary graphs to the set defined by three degrees of freedom.

- Size of the graph, defined by the number of vertices [n]
- Functional class, defined by cardinality histogram [T1-T6]
- Uniformity of street lengths, defined by a variance figure [σ]

This still results in a large number of possible graph setups. Before evaluating all algorithms on all possible graph setups we quantify how much the different degrees of freedom influence the objective function in terms of convergence behavior. We also research their influence on the resulting model in terms of systematic characteristics in the weight distribution of the fitted models. Since we are focusing on the effect of different set ups instead of different optimization techniques, we use the L-BFGS algorithm 5.2.1, which is not efficient for all scenarios but is guaranteed to converge.

Table 6.4: Simulation parameter setup

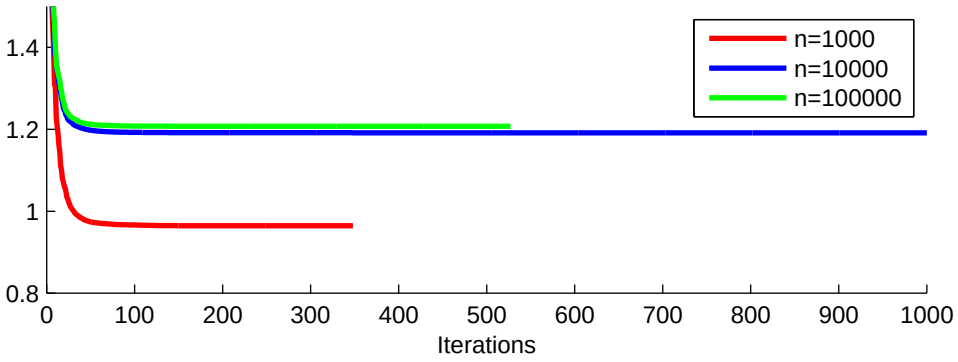
Simulated graph	
Topology setup	$\{T1\text{-}T6\}$
Number of vertices	$\{1000, 10000, 100000\}$
Street length distribution (σ)	{1.0,5.0}
Weight distributions	$W1$
Optimization	
Objective function	$O0$
Algorithm	L-BFGS,SGD
Linesearch	L-BFGS with Wolfe-Backtracking
Start-values	$N(0, 1)$

Results

We run the LBFGS on all set ups mentioned above with uniformly distributed feature weights as described in section 6.1.1. The complete result set can be found in the appendix 9.1. To summarize those results we present here the effects of the degrees of freedom mentioned above:

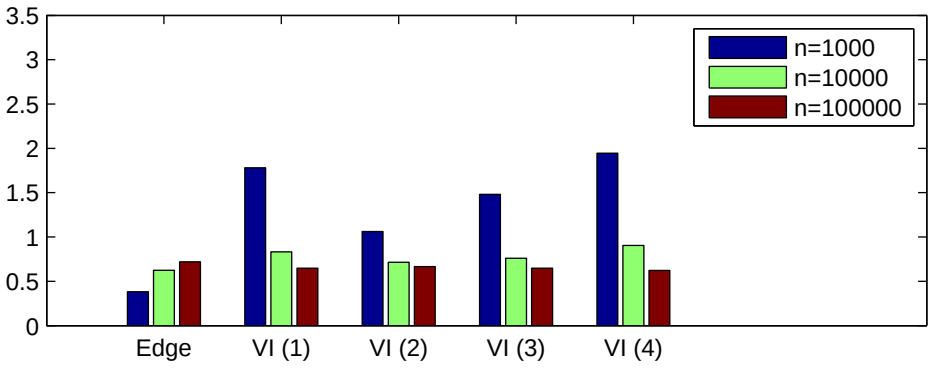
- Size of the graph (n):
  In our simulations graphs of size $n = 10000$ always converged slowest. The fast convergence of small graphs ($n = 1000$) stems from the fact that many features are constant zero, since the specific realization of the indicator functions are not observed.
  The weight distribution is irregular for low n. High weights can be observed for one-

Figure 6.7: Convergence speed example - Topology T2

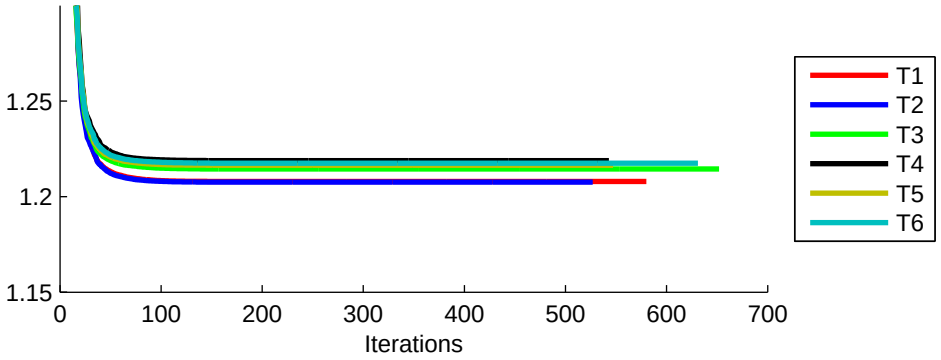


dimensional indicator functions. Further we notice that the weights are uniformly distributed for n that resemble real life applications ($n > 100000$)

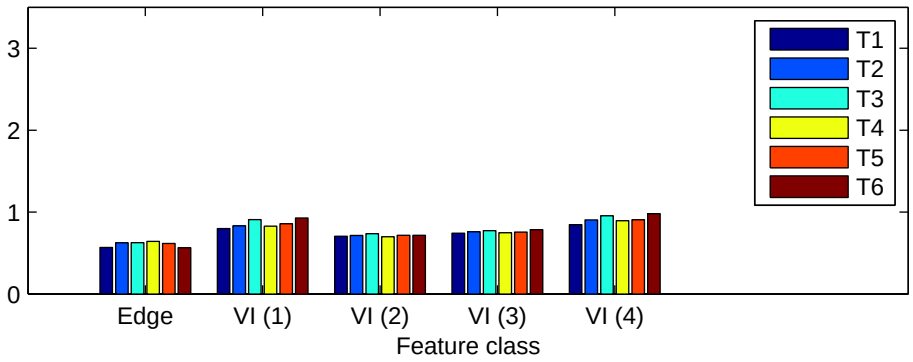
Figure 6.8: Weight Distribution Histogram - Topology T2



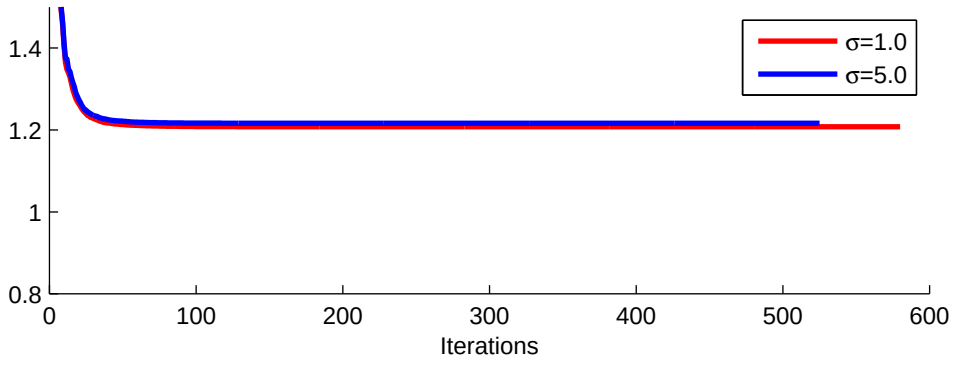
- Functional class(T1-T6):
  Convergence is slightly slower for graphs with a higher amount of cardinalities > 2. The effect however is relatively small. The overall fit is also slightly better (lower neg-log-likelihood) for T1 and T2. This however is not essential for the convergence behaviour.



  The weight distribution does not systematically depend on the histogram of cardinalities and therefore on the functional class of the street graph. The trend observed for smaller to bigger graphs is the same for all classes.



- Street length distribution:
  We recognize more evenly distributed street networks ($\sigma = 1.0$) to have lower neg-log-likelihood values. However, there is no faster convergence or effect on the resulting weight distributions.



Conclusion:
The three degrees of freedom can be reduced down to the shear size of the graphical network. There were no significant dependencies on the functional class or the street lengths distribution. In the evaluation of optimization methods we use the following reduced set of set ups:

- Size: $n \in 1000, 10000, 100000$
- Functional class: T1 (since it resembles motorways, most important to VMC)
- Street length distribution: $\sigma = 1.0$

6.2.2 Feature weighting

Analogously to the influence of the topology setup in 6.2.1 we now analyze the influence of the different weighting scenarios (W1-W4, as in 6.1) to the convergence behavior of the LBFGS on the log-pseudolikelihood. Using no penalty terms in the objective function and no model selection algorithmics we expect the effect of different weighting scenarios to be neglectable. The simulations are conducted in order to confirm that general assumption for LBFGS optimization. For the simulations we chose the graph setup that was described in the previous section. Therefore the paramters for the simulation set ups are:

Table 6.5: Simulation parameter setup

Simulated graph	
Topology setup	$T1$
Number of vertices	$\{1000, 10000, 100000\}$
Street length distribution (σ)	1.0
Weight distributions	$\{W1, \ldots, W_4\}$
Optimization	
Objective function	$\{O1, O2\}$
Algorithm	L-BFGS
Linesearch	Wolfe-Backtracking, fixed parameters (c1,c2,minStep,...)
Start-values	$N(0, 1)$

Results

The resulting convergence graphs can be found in the appendix 9.2. An exemplary result is also given in Figure 6.9.

Figure 6.9: Topology: T1, $n = 10000$, $\sigma = 1.0$, no penalty used

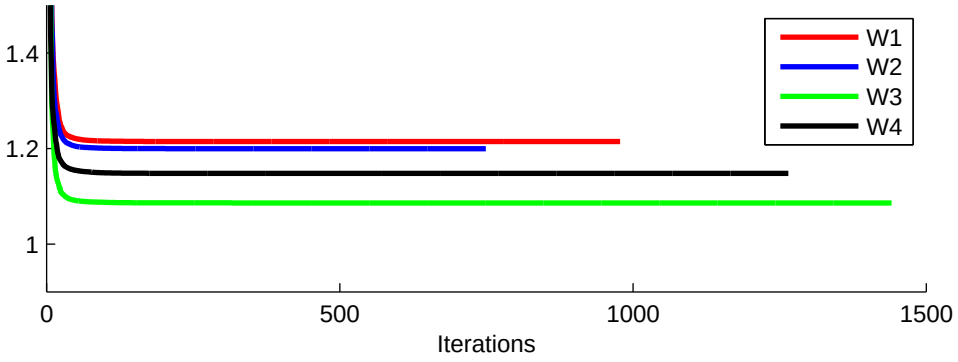
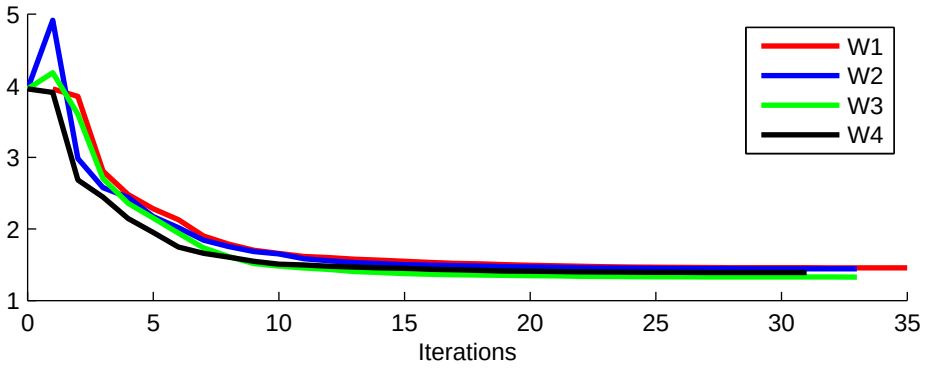


Figure 6.10: Topology: T1, $n = 10000$, $\sigma = 1.0$,with L1-penalty term



Two distinct observations can be made from this set of simulations

- The weighting is not essential for the convergence speed for the negative log-pseudolikelihood (O1)
- Also, when using penalization with the L1-penalty term, there is no measureable performance advantage in the W2 and W4 scenario, due to the high probability of low weights. The actual processor-runtime can however be reduced by using thresholding in all weighting scenarios.

Conclusion:
We ommit the different weighting scenarios in simulations without penalty terms (Objective function O1) but use all variants in simulations with penalty term (Objective function O2-O5).

6.2.3 Stochastic Gradient

To compare the efficiency of deterministic approaches and stochastic approaches we use the number of 'effective iterations' as introduced in section 6.1.5 instead of the number of iterations itself. This accounts for the fact that one iteration in deterministic approaches is far more costly and that iterations are therefore not equivalent to computation time in this context. We are using the LBFGS as method of choice for the deterministic benchmark. It uses a Wolfe backtracking linesearch algorithm.

Setup

In this study we are using a simple stochastic gradient descent algorithm (as in 5.3.1) with different Robbins-Monro decays as step lengths, as well as fixed step sizes. The mini-batches used for the stochastic approach is fixed at size 100. The influence of the mini-batch size is also shown in figure 6.13.

Table 6.6: Simulation parameter setup

Simulated graph	
Topology setup	$T1$
Number of vertices	1000
Street length distribution (σ)	1.0
Weight distributions	$W1$
Optimization	
Objective function	$O1$
Algorithm	L-BFGS,SGD
Linesearch	L-BFGS with Wolfe-Backtr.,
	fixed params.(c1,c2,minStep,...)
	SGD Robbins-Monro, decay param. $\alpha \in \{0.5, 0.75, 1.0\}$
	SGD with fixed steps $\delta \in \{0.001, 0.01, 0.1\}$
Batchsizes	{5,10,50,100}
Start-values	$N(0, 1)$

Results

The full set of results can be found in the appendix in section 9.3.

Figure 6.11: L-BFGS vs. SGD with RM-Decay, with mini-batches of size 100

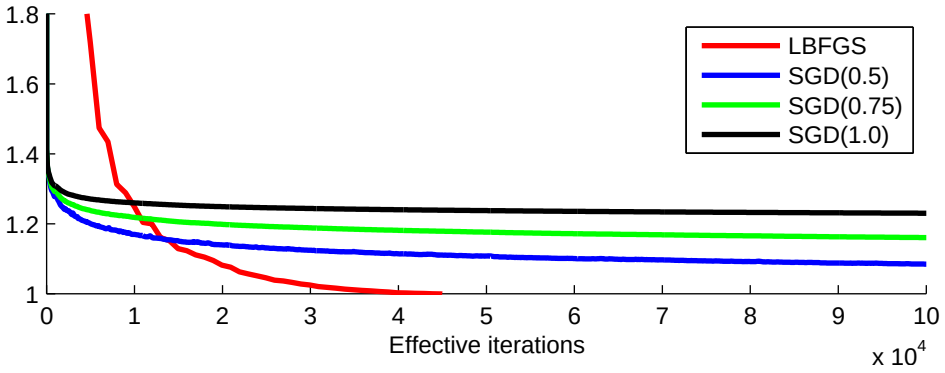
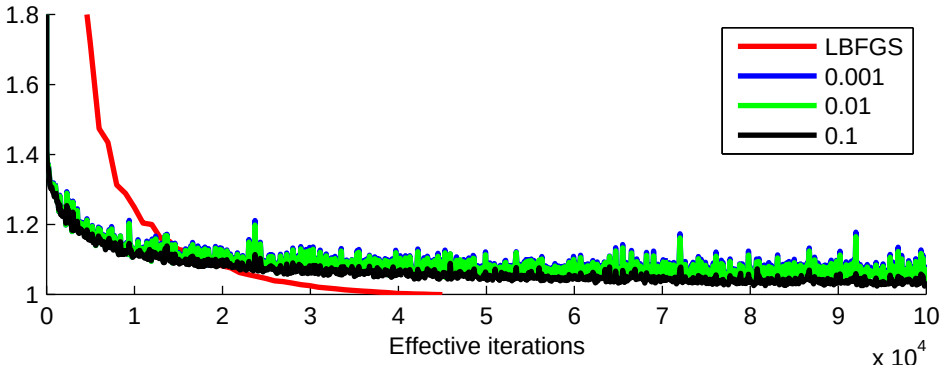


Figure 6.11 shows that the stochastic gradient outperforms the LBFGS up to a certain break even point. However using Robbins-Monro based stepsize decay results in a very slow convergence behaviour if the start values are picked poorly or the mini-batch is chosen to be small. Figure 6.12 shows the same stochastic gradient descent with a fixed step size. This has similar properties as the decay strategy based approach and is also weaker than LBFGS after the break even point. The jittery curve form results from the stochastic nature of the objective function. This can be smoothed out by filtering afterwards or by using the Stochastic Average methods described in 5.4.

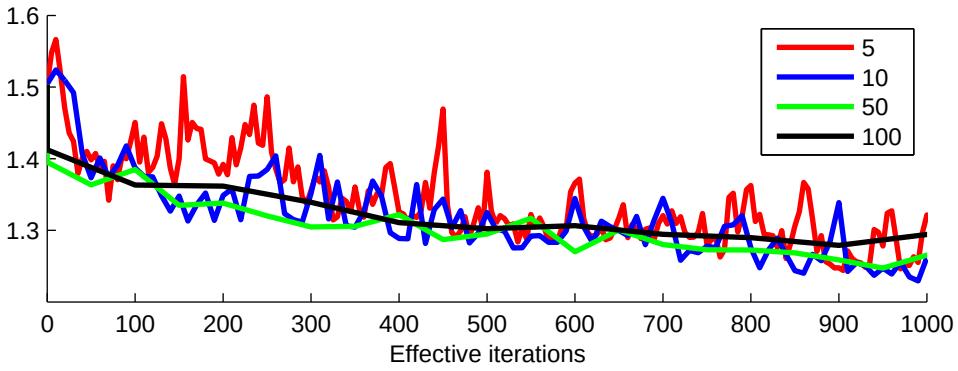
Figure 6.12: L-BFGS vs. SGD of constant step size, with mini-batches of size 100



Using lower mini-batch sizes results in more noise as can be seen in figure 6.13. On the other hand it also shows that in terms of effective iterations, using big batch sizes is not guaranteed to perform best. Defining an optimal batch size is highly dependent on the optimization problem and is not researched here.

- Effective iterations are far less in stochastic approaches

Figure 6.13: SGD of constant step size, with different mini-batche sizes



- Since the starting values is not too far off, large decays converge faster (choosing big offsets changes that behaviour)
- Choosing bigger mini-batches is not beneficial (if processed serialized)
- Choosing bigger mini-batches absolutely beneficial (if processed paralelized, looking at iterations instead of effective iterations)

6.2.4 Stochastic Average Gradient

We introduced SAG in section 5.4. The idea is to use the average over past local gradient evaluations in a stochastic optimization in order to receive a more accurate approximation of the objective function. We introduced a ringbuffer implementation of the SAG, which solves the problem of a high space complexity for storing past gradient information per vertex. One major benefit of SAG is that it converges with a constant step-size α, if α is chosen small enough (as described in [RSB12]). In this section we compare the ringbuffer implementation to the standard Stochastic Gradient Descent.

Setup

In this study we are using a simple Stochastic Gradient Descent algorithm (as in 5.3.1) and Stochastic Average Gradient (as in 5.4) with different fixed step sizes.

Table 6.7: Simulation parameter setup

Simulated graph	
Topology setup	$T1$
Number of vertices	{10000}
Street length distribution (σ)	1.0
Weight distributions	$W1$
Optimization	
Objective function	$O1$
Algorithm	SAG with Ringbuffer (SAG)
	SAG with weighting: (SAG-W1)
	SAG with weighting:(SAG-W2)
	SAG with local weighting (5.4.4) (SAG-LW)
	SAG with robust weighting (5.4.5) (SAG-RW)
	SAG with robust local weighting (5.4.6)(SAG-RLW)
Linesearch	Constant Step Sizes: {0.01,0.1,1}
Batchsize	{10,50,100}
Start-values	$N(0, 1)$

Results

Figure 6.14: Step size: $\delta = 0.01$

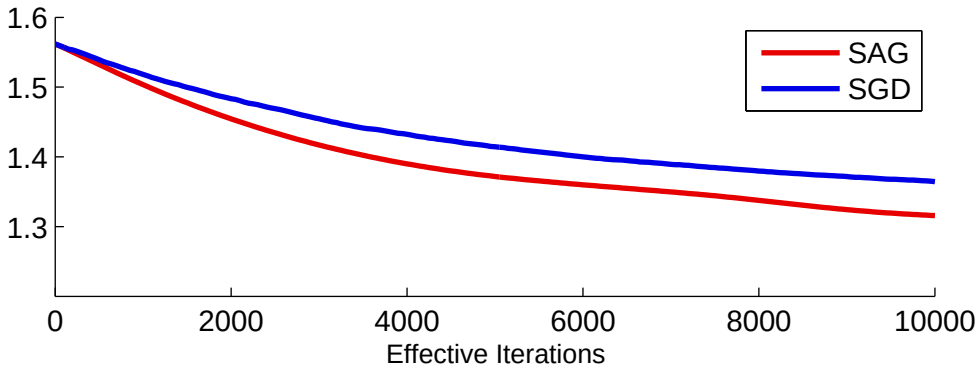
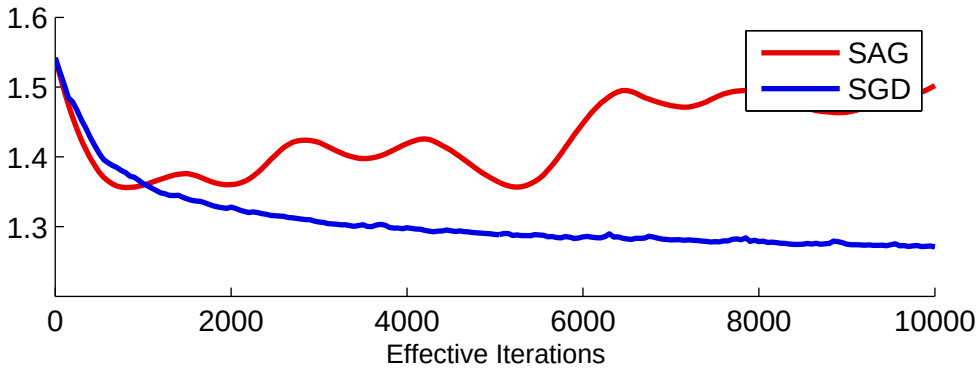


Figure 6.15: Step size: $\delta = 0.1$



- SAG performs better in all cases where convergence is reached
- Due to the momentum effect (see 5.5) SAG diverges when using higher stepsizes (boundary 0.03125)

6.2.5 Stochastic Average Gradient Variants

Stochastic average gradient methods guarantee convergence for stepsizes below the certain limits. For large number of samples, or in CRF scenarios, large number of vertices Roux et al. [RSB12] use the upper bound of

$$\alpha \leq \frac{1}{16L} \quad \text{L being the Lipschitz constant of } g$$

as a boundary, independent of the degree of convexity or the size of the problem. However in their paper they already propose that for specific problems this boundary might not be sharp. Therefore we conduct a study on how different constant step sizes turn out in training CRFs with the different Stochastic Average Variants.

This set of simulation is used to show a qualitative comparison of these SAG variants in terms of convergence behaviour dependent on

1. The line search step size
2. The batch size
3. The weighting parameters

Setup

Table 6.8: Simulation parameter setup

Simulated graph	
Topology setup	$T1$
Number of vertices	$\{1000, 10000, 100000\}$
Street length distribution (σ)	1.0
Weight distributions	$W1$
Optimization	
Objective function	$O1$
Algorithm	SAG with robust local weighting (5.4.6)(SAG-RLW)
Linesearch	Fixed step sizes $\{0.1, 0.2, 0.5, 1, 2\}$
Batchsize	{10,50,100}
Start-values	$N(0, 1)$

Results

The complete results can be found in the appendix (9.5).

Conclusion:

The theoretical boundary $\alpha \leq c = \frac{1}{16L} = \frac{1}{32}$ is not sharp in the CRF case and highly dependent on the model as well as the SAG variant.

Figure 6.16: Step size: $\delta = 0.01$

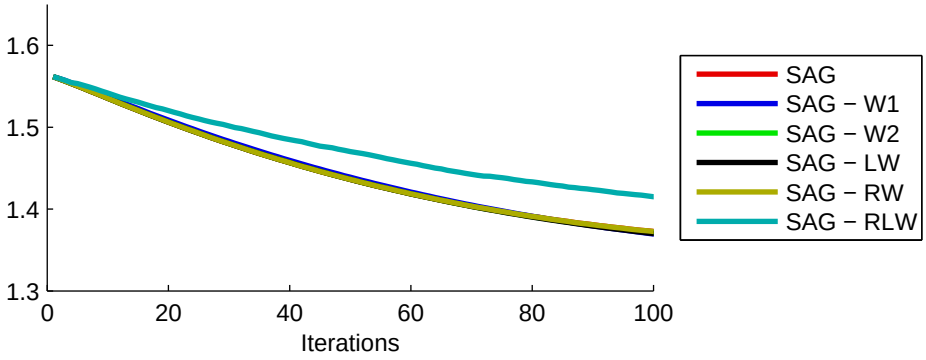


Figure 6.17: Step size: $\delta = 0.1$

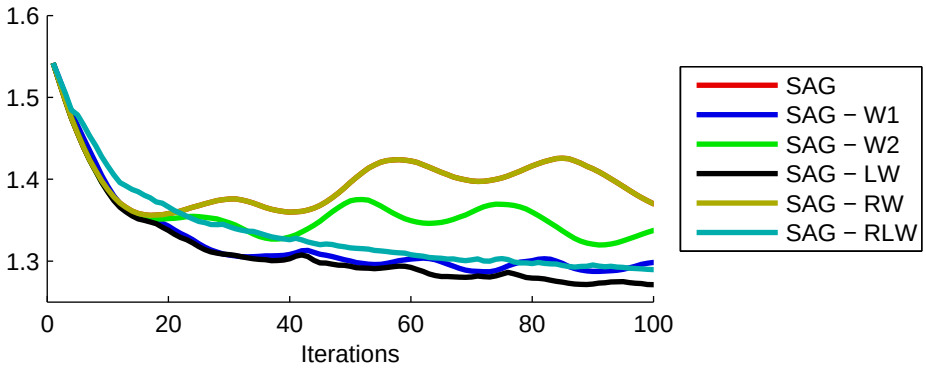
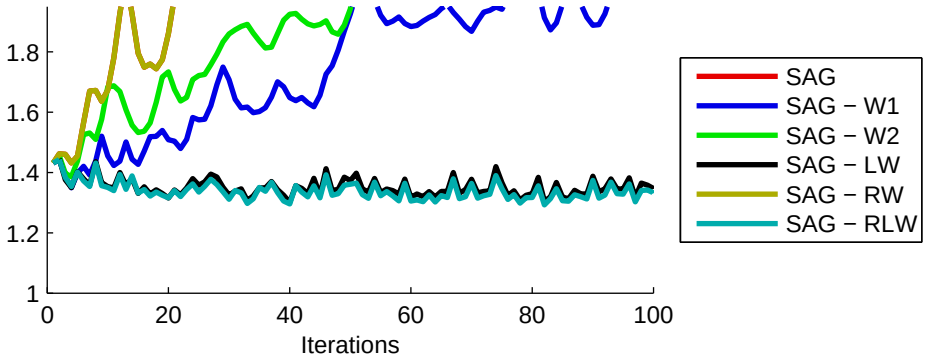


Figure 6.18: Step size: $\delta = 1$



We observe the following behaviour:

1. Using small step sizes (< 0.03125) all variants converge

2. Using bigger step sizes the locally weighted (SAG-LW) and (SAG-RLW) are the only variants to converge
3. The convergence speed highly increases with higher step-size values, therefore SAG-LW and SAG-RLW are regarded as beeing the be most effective
4. The results do not depend on the weighting parameters of the different SAG variants

6.2.6 Local Robust Stochastic Average Gradient

Stochastic Average Gradient with local robust weighting converges with rather big step sizes and therefore outperforms standard SAG and SGD methods. Further it is independent of the parameter space scale as described in 5.4.6. The free weighting parameter k specifying the width of the kernel by the distance to the k-th neihgbour in the ringbuffer is not as significant as the weighting parameters for the other SAG variants. When using reasonable batch sizes of > 10 and rinbuffers of size > 100, which is highly advised, than the values of the k nearest neighbours around the last position in parameter space. This results in almost exactly the same behaviour for $1 \leq k \leq 10$. The evaluation of the SAG-RLW optimization will therefore concentrate on empirically finding an optimal step size for the different topology sizes.

Setup

Table 6.9: Simulation parameter setup

Simulated graph	
Topology setup	$T1$
Number of vertices	$\{1000, 10000, 100000\}$
Street length distribution (σ)	1.0
Weight distributions	$W1$
Optimization	
Objective function	$O1$
Algorithm	SAG with robust local weighting (5.4.6)(SAG-RLW)
Linesearch	Fixed step sizes 0.1,0.2,0.5,1,2
Start-values	$N(0, 1)$

Results

The complete results can be found in the appendix (9.6).
Conclusion:

We observe the following behaviour concerning step sizes:

- Convergence behaviour is better for bigger mini-batch sizes, by a large degree
- Convergence behaviour is better for smaller problem sizes, by a very small degree.
- In all simulated scenarios a step size of 0.5 had a good convergence behaviour.

Figure 6.19: Number of vertices $n = 1000$, SAG-RLW, Batch size 10

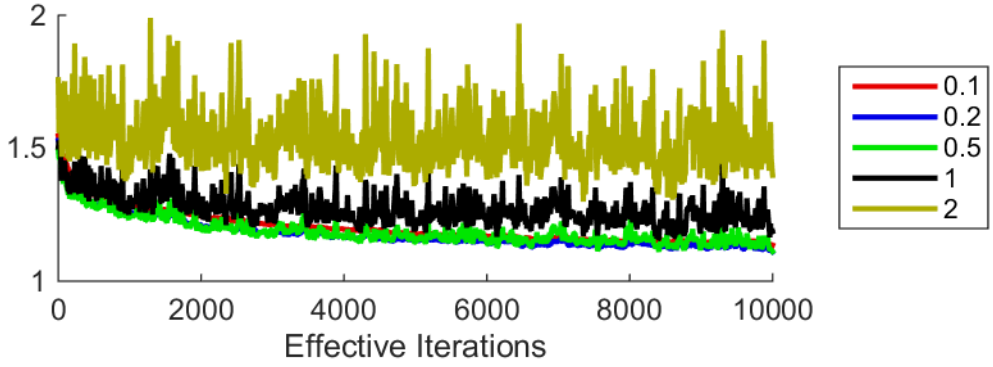


Figure 6.20: Number of vertices $n = 10000$, SAG-RLW, Batch size 50

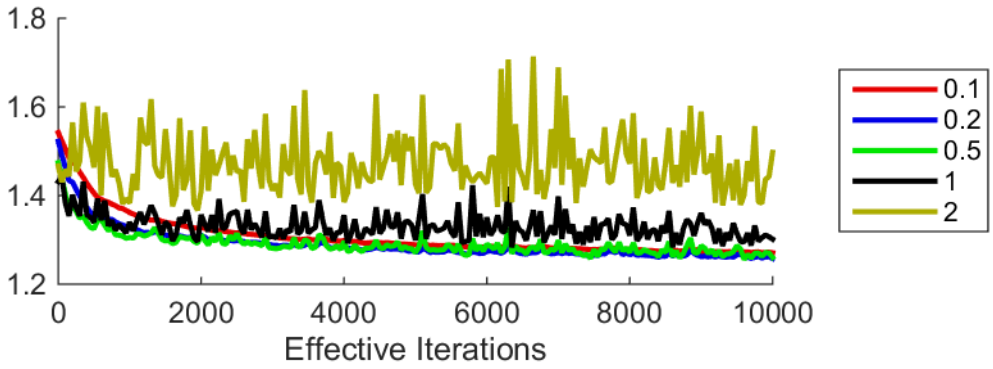
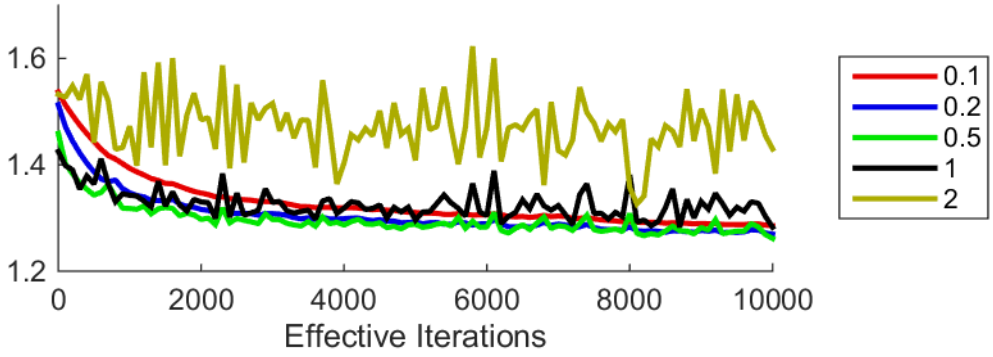


Figure 6.21: Number of vertices $n = 100000$, SAG-RLW, Batch size 100



6.3 Conclusions

The artificial street graph setups we use were investigated. The influence of the setup parameters on the convergence of BFGS and distribution of weights among the features. Our simulations show that there is only very little difference in the setups and that the most important parameters is the network size, or the number of vertices in the graphical model. Therefore we limited the degrees of freedom for the optimization simulations to the number of vertices.

Using conditional random fields with a single feature template assumes that the random field distribution is spatially self-similar. This encourages the use of stochastic optimization algorithms. In many fields the most efficient deterministic approach is L-BFGS which shows good convergence behavior but is of high complexity due to the evaluation of the full likelihood function. As shown in the above section in the artificial set ups that we design in order to benchmark the introduced algorithms, L-BFGS performs worse than simple SGD and SAG algorithms. The field of stochastic optimization is very wide and we chose to focus on the stochastic average gradient methods. The SAG itself is of high space complexity, and we offer a solution by introducing a memory-limited ringbuffer version. The theoretical asymptotic properties of the SAG stay the same. However we also introduce modifications to the SAG by defining different strategies of weighting past gradient contributions. Besides faster convergence for a fixed, chosen stepsize they also proof to be more robust when the theoretical step size boundaries which ensure convergence are disregarded. This leads to a boost in convergence speed in practical applications. We therefore propose the following procedure for determining a strategy for efficient training of CRFs:

- Determine the problems Lipschitz constant L
- Use SAG with robust-local-weighting and stepsize $\alpha = c \cdot \frac{2}{L}$ with free parameter $c > 0$
- In our simulations $c > 2.0$ was valid in almost all cases, higher values are possible for different scenarios
- Monitor convergence to detect non-convergence

If the complexity of the problem is rather low ($\dim(\boldsymbol{\theta}) < 10^3$ and $|V| < 10^4$) we still advise to use deterministic approaches, for a stochastic optimization techniques can by definition only converge to a value in a ϵ ball around the optimum.

7 Application

So far we have introduced CRFs as a suitable model for road quality measurements. We discussed and evaluated different techniques for inference and training. In this chapter we focus on the application and the application based questions regarding road quality models. We start in section 7.1 with the definition of typical application (and VMC) related extrapolation tasks and the specific demands for model training and inference. In section 7.2 we discuss the necessary preparation steps in terms of data preprocessing and study preparations. Afterwards we evaluate the use of CRFs performance on the datasets of Sweden (in 7.3) and Finland (in 7.4). These studies are also part of an unpublished report [SL$^+$14] which was written in the context of the Virtual Measurement Campaign project at the Fraunhofer ITWM. We finish this chapter by giving an overview of how CRF extrapolation can be used to transfer data within the environment database of the Virtual Measurement Campaign.

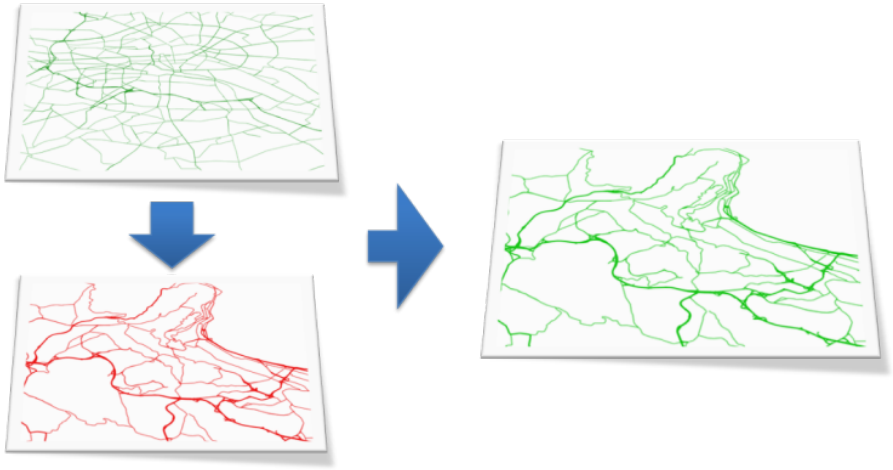
7.1 Extrapolation tasks

The following extrapolation tasks are generalized tasks that arise when dealing with street quality and vehicle damage related problems. They do however easily extend to other fields of modeling where graphical models are used for extrapolation. The different extrapolation scenarios are motivated by application oriented tasks. They differ in the definition of training and testing dataset, however they can all be handled by training CRFs with the techniques described in chapter 4 and 5.

7.1.1 Independent Extrapolation

We assume that training and test dataset are statistically independent, due to a non connected topological structure. This is the classical classification problem identically to most classification problems in image recognition or speech related tasks. One major difficulty in this scenario is a possible overfitting of the model parameters. CRFs as a maximum entropy model limit this problem and by reducing the parameter space by penalized model training the overfitting effect is further reduced. An example for independent extrapolation within the Virtual Measurement Campaign is illustrated in Figure 7.1. This represents use case where a user wants to extrapolate information of one region to another which is not overlapping or intersecting with the first.

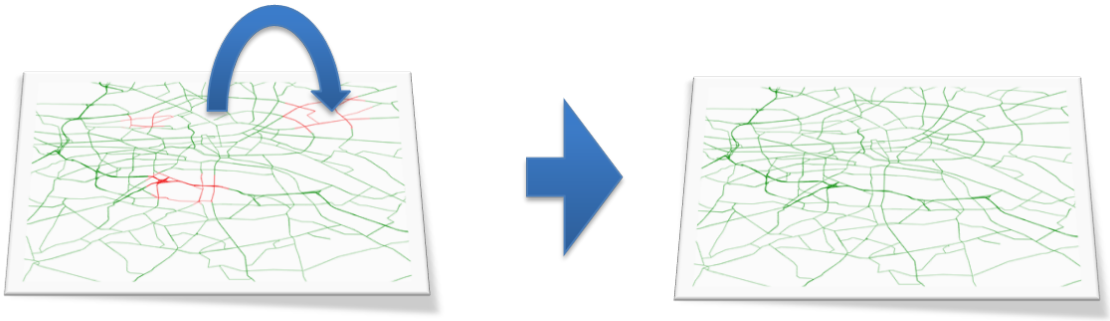
Figure 7.1: Extrapolation of road related information



7.1.2 Filling Gaps

If the training dataset is much bigger than the test dataset and the sets are interconnected we will refer to the testdata as target information gaps. Classification on the test dataset is equivalent to the filling of the label gaps in the complete dataset. There are two majorly different approaches to approach the classification problem. If the gaps in target data are large or contain a complex topology, we have to make use of latent variables and train the model with algorithms that can deal with latent variables. In many cases these are variants of expectation maximization algorithms, which are usually of higher computational complexity than the training methods we discussed in chapter 5. If the gaps are small we can use standard training methods. In the VMC context this kind of extrapolation can be used to fill in missing data of small portions of the street network in order to obtain a 'complete' dataset. Such a use case is depicted in Figure 7.2.

Figure 7.2: Filling information gaps with CRF based extrapolation



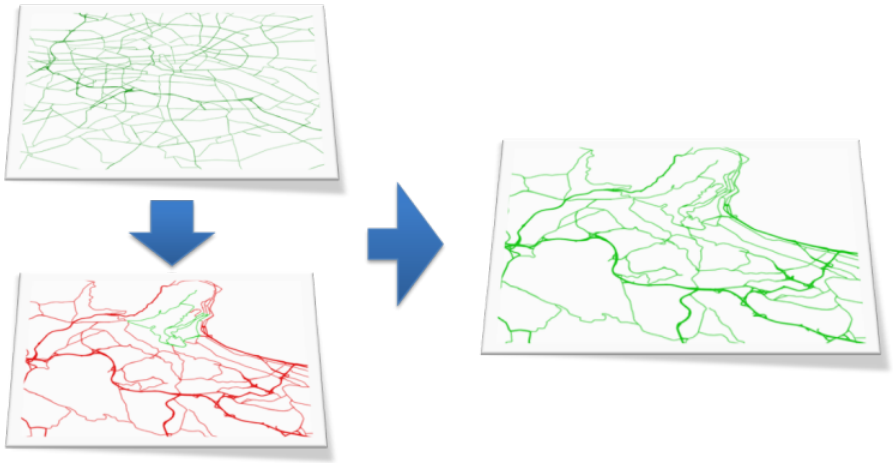
7.1.3 Extrapolating Exclaves

Another application for extrapolation is a modification of the independent extrapolation scenario. Imagine the case where a user wants to gain information about the road quality of a certain region by training the model with data that is split into a part of high relevance and a part of low relevance. The most extreme case would be training a model that is party located within the region of interest. Such a case is depicted in Figure 7.3. A prototypical scenario would be to model for example Japanese road qualities by training with a big dataset of German measurements and a small exemplary measurement situated in Japan itself. This is highly relevant for practical applications, since a small measurement campaign in the region of interest can be used to increase the data quality immensely. Such a scenario raises two questions.

1. Is the training data to be treated equivalent or weighted by region?
   Weighting the training data can be achieved by adding each of the two different parts of the training set a different amount of times into the training set for the optimization. It is also possible to randomly choose subsets from the different regions. However using stochastic optimization on an artificial problem with multiple graph copys of the two regions is effectively the same. In pratical application the weighting can be treated as a user defined preprocessing parameter.

2. How is the data of the target region incorporated in inference?
   Inferring from the model can be achieved by using standard MCMC with Gibbs sampling (3.3).

Figure 7.3: Extrapolation information in areas of little information



7.2 Data Preprocessing

7.2.1 Segmentation

Data segmentation describes the process of defining entities that hold information and serve as nodes in the graphical model. In the road quality context this is equivalent to the task of finding a suitable length of road that is regarded as having a constant vector of label data (road quality) and observations data (traffic,etc. . .). Most road roughness data is collected at a very high spatial resolution, since it is a result of very high frequency measurement, such as a 1-KhZ lidar. This data is used to create a road profile (see 1.2.1) which again are used to calculate roughness indicators (see 1.2.2) in the form of summary statistics. Most publicly available datasets offer these indices for segments of standardized length. The German BASt and the Finish Road Administration use a standard segment length of 100m, while Sweden offers with 20m a higher measurement granularity. However we accumulate those segments to 100m segments in order to be comparable to Germany and Finland. Therefore the 100m segmentation can be regarded as a lower bound for the road segmentation in our practical application. This segmentation is also used for the statistical models of road profiles by [JR13] introduced in 1.2.6.
With the 100m segments as minimal segmentation level, the question remains if and how segments should be aggregated to bigger segments. These further accumulations influence the topology of the graph of the road network and therefore also have an effect on the optimal feature weights of a fitted CRF. This is representing the Markovian nature of the measurements, which have shown in the prestudy on the finnish road data (1.4). This study was based on 100m segments. Accumulating these segments to larger ones results in weaker neighborhood dependencies. In the VMC software (see 1.3) the segmentation is determined by a set of traffic relevant observable observations. These are, number of lanes, street name (or number), speed limit and the functional class. Topologically connected segments are accumulated as long as these figures are invariant. The data that is used in this chapter is not taken directly from the VMC database but the original data from the Swedish and Finnish authorities and therefore not preprocessed in terms of segmentation. The data used in this study is organized in tabular form as prototypically depicted in Table 7.1. Similar to the VMC segmentation we accumulated the 100m road segments from Table 7.1 to longer segments, by grouping connected segments that have the exact same observable variable realizations (Functional class, Speed limit, ADT, . . .). That way the original dataset changes in terms of the number of vertices and the vertex cardinality distribution.

7.2.2 Preparing observations

Censoring data

Dealing with observation figures of continuous nature usually requires some preparation. We discussed the necessary step of normalization of the features in 2.5.2, which is a

Road Number	Lane	Length	Measured IRI	Functional class	Speed limit	ADT
1	1	100	2,1	1	60	41308
1	1	100	1,2	1	60	41308
⋮						⋮

Table 7.1: Excerpt from Finnish Road Roughness Data

numerically motivated step. Since this is an inevitable step if numeric sparsening is used, it is not useful to previously apply any transformation that could be nullified by normalizing the feature. Especially when we want to avoid information loss in the normalization, a very valuable preprocessing step for continuous or high dimensional discrete values is the elimination of outliers and false data. This can vastly increase the quality of the features relying on the continuous observations since it limits the risk of clumped up data after feature normalization. Essentially the task of eliminating outliers is to find a suitable set of bounds between which the data is declared valid. Since this task is strongly related to feature normalization it shares the same difficulties when trying to find suitable bounds from the dataset. The most convenient solution is to define these bounds manually. That way it is ensured that the data is not misinterpreted.
If that is not possible or feasible the bounds have to be determined automatically. In general it is advised to use the test and the training data to find suitable bounds. Taking the empirical bounds for the observation from the test dataset alone and censoring the training data from the training dataset accordingly leads to a model training is highly fit to the test datasets, and risks overfitting.

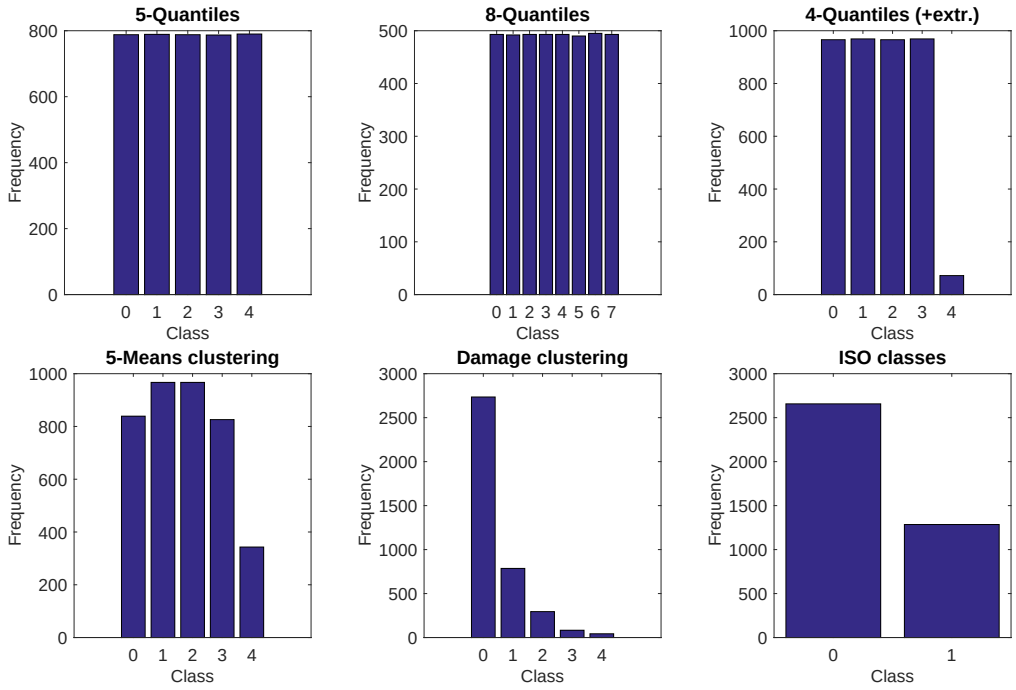
Defining classes

As a classification technique, modeling via conditional random fields, requires a target figure with values in a finite set $\mathcal{Y}$. Whenever the data basis for the target figure delivers continuous values or the set of values is finite but is considered too big (i.e. $|\mathcal{Y}| \gg 10$), we discretize these values by categorizing. For example, in many datasets the IRI is given as continuous figure. There are several plausible ways to discretize the IRI. In their paper [JR13] Johannesson and Rychlik use bins of sizes of increasing powers of 2. We will refer to that binning as 'ISO classification', since it is motivated by the ISO definition of the IRI. Besides this rather physically motivated discretization we also consider binning by empirical quantiles. This ensures an optimal distribution of samples regarding the target figure. In general, the choice of class boundaries is essential and can be motivated from essentially these two perspectives.

1. Statistically it is beneficial to use empirical quantiles or clustering to achieve evenly distributed classified observations.

2. On the other hand it can be useful to use content based class boundaries. This can lead to much better interpretation.

Figure 7.4: Histograms for different IRI classifications



For road quality datasets it is usually beneficial to define one class of observations that covers extreme observations (like very high average speeds) and use quantile or cluster driven class bounds for all other classes. These classification principles are also applicable for continuous labels. The international roughness index for example is often measured as a value on a continuous scale. In order to define a classification problem we use one of the above methods to classify the target quantity. The resulting categorial distributions on the Finnish dataset are depicted in the histograms in Figure 7.4. The ISO classification is not well suited for distinguishing the road quality in the datasets we use in this chapter (Finland and Sweden) due to the fact, that all roads belong to the first two of eight defined classes. The roads in this dataset are major roads and motorways and therefore of good quality in terms of roughness. This leads to this condensed spectrum of ISO defined classes. The quantile based classification is equally distributed by definition. The extreme value class we defined uses a the boxplot definition of outliers for the class boundaries. The k-means approaches use a standard clustering algorithms to determine 5 clusters. In the original definition the absolute value is being used a metric on the IRI values. In the damage oriented version we use $d(x, y) = |x^5 - y^5|$ as metric, for this is a distance measure for the expected inflicted damage as described in [JPR14].

7.3 Sweden road quality

7.3.1 Data source and preparation

For this study we combine three sources of data. The first is a dataset collected and offered by the Swedish Road Administration (Trafikverket, 2011). It contains information describing the immediate condition of the road by using figures like the international roughness index (IRI), the pavement width, annual daily traffic (ADT) and others. The individual road segments are given in driving direction and can be associated with the street names. This allows the construction of a linear graphical model. In addition, the dataset holds information about the administrative region the street segments are located in. This allows us to localize the street segments in other datasets. The second dataset holds socioeconomic information more coarse level (NUTS 3) Eurostat offers a dataset of different socioeconomic figures which can be projected onto the areas, that are known for each street segment. In a similar way, we can make use of the regional analysis output from the VMC software. Such regional analysis results contain information like curviness and hilliness of the streets.

As mentioned above the data is preprocessed and outliers are detected. We use the standard boxplot definition for outliers. Values greater than the 75% quantile plus 1.5 times the interquartile range are defined as outliers. The equivalent for low valued outliers. We omit these data points in further analysis. In this particular dataset there are 9809 outliers among 268224 road segments, a rate of 3.7%. Additionally we filter the dataset of obvious errors, and use only the data for one lane in one direction, since these are minimal characteristics fulfilled by all street segments within the dataset.

As described above, the streets are divided into 100m segments. We accumulate segments to bigger 'road' objects by grouping connected segments that belong to the same road (identified by the road number or name) and that are associated with the same administrative region on the LAU 1 (Eurostat definition) level. This corresponds to a subdivision of the roads of sweden by the borders of the municipalities they are passing through. This way we receive a reduced number of 'road' entities which can be uniquely associated with a region. These regions can be used to add information from the VMC database (hilliness and curviness), as well as socioeconomic data from Eurostat. This segmentation leads to 3942 road data points in total.

7.3.2 Linear Model

As a reference model for the evaluation of CRFs as extrapolation model we choose linear regression models. Linear regression models are of the form:

$$Y = \boldsymbol{\theta}'\mathbf{X} + b$$

with Y again denoting the label variable and X the vector valued observation variable with the explicit constant offset $b \in \mathbb{R}$ and $\boldsymbol{\theta}$, the vector-valued model parameter. The

Table 7.2: Trained linear model parameter

$\boldsymbol{\theta}_i$	Estimate	Observation	p-Value
1	0.5412	constant	5.34E-05
2	0.1479	Road Category	4.34E-16
3	0.0737	Functional Class	0.0029453
4	-0.0852	Curviness is low	0.45278
5	0.0069	Curviness is moderate	0.95374
6	0.0189	Curviness is high	0.87584
7	0.0038	Hilliness is low	0.97384
8	0.0021	Hilliness is moderate	0.98572
9	0.0275	Hilliness is high	0.81476
10	-0.0023	Speed Limit	1.88E-09
11	-0.0895	Pavement Width	5.17E-163
12	0.011	Pavement Age	1.63E-80
13	-0.0002	Population Density	0.04281
14	2.4122	Employment Rate	6.48E-22
15	-4.616	Self Employment Rate	5.96E-15

main disadvantage of the linear model compared to the CRF model is the fact that the topological information can not be used in the model training. On the other hand it is much easier to train the model parameter and penalization is very well understood. Motivated by the lognormal nature of the IRI (see 1.4), we fit the following model to the dataset:

$$\log IRI = \theta_0 + \boldsymbol{\theta}'\mathbf{X}$$

Where θ_0 denotes an offset and the design matrix X holds dummy variables for the segment-wise discrete observations:
{ road category, functional class, curviness, hilliness }
as well as the continuous figures that are used as regressors in the model:
{ speed limit, pavement width, population density, employment, self employment rate}

7.3.3 Model comparison

We compare the CRF modeling to standard linear modeling techniques. In order to measure the model quality when used as a predictor in an extrapolation task, we need to define a set of measures which are applicable for both models. On the one hand we are interested in extrapolating the IRI as a representative for road roughness, since it is widely available in datasets and internationally used for describing road quality. Another, more VMC specific quantity is the expected damage indicator described by Johannesson and Rychlik. We propose to use the prediction error regarding those two quantities as quality measure for the introduced models. The prediction task usually requires a training and a testing dataset, where we train the model using the data of the first and predict

the target quantities of the second dataset. We construct these datasets by omitting a random subsample of the complete Swedish dataset and use this as a pair of training and testing data as depicted in Table 7.3.3. This procedure is done several (10) times in order to avoid random effects, specific to the choice of omitted roads.

	Training data	Test data
Leave 1% out	99% of Swedish roads	Remaining 1% of Swedish roads
Leave 10% out	90% of Swedish roads	Remaining 10% of Swedish roads

Let $\hat{y}_k$ denote the prediction of the k-th street of the unknown streets (1This estimator can be realized by using the linear model or the CRF alike. We now use the following measures to compare the models:

Bias estimator for the IRI :	$\text{BIAS}_{\text{IRI}} = \frac{1}{N}\sum_{n=1}^{N}(\hat{y}_k - y_k)$
Mean square error of the IRI :	$\text{RMSE}_{\text{IRI}} = \sqrt{\frac{1}{N}\sum_{n=1}^{N}(\hat{y}_k - y_k)^2}$
Bias estimator for the expected damage :	$\text{BIAS}_{\text{ED}} = \frac{1}{N}\sum_{n=1}^{N}(\hat{y}_k^5 - y_k^5)$
Mean square error of the expected damage :	$\text{RMSE}_{\text{ED}} = \sqrt{\frac{1}{N}\sum_{n=1}^{N}(\hat{y}_k^5 - y_k^5)^2}$

7.3.4 Results

Table 7.3: Extrapolation results using CRF and linear model

		CRF		Linear Model		Reduced Linear Model	
		RMSE	BIAS	RMSE	BIAS	RMSE	BIAS
Leave 1% out	IRI	0.78	-0.08	0.64	-0.08	0.67	-0.08
	Damage	196.45	-46.5	364	-177	388	-187
Leave 10% out	IRI	0.77	-0.06	0.62	-0.1	0.66	-0.09
	Damage	194.16	-105	378	-184	409	-192

We recognize three major results in this study.

The linear model performs slightly better in predicting the IRI while having a higher error when used as a predictor for the damage. This results from the fact that the linear model is optimized to predict log-IRI data, since that is observed to be almost normal distribution. On the other hand the discretization, as a preprocessing step in CRF modeling can be tuned in order to perform better on IRI, instead of the damage.

The damage estimation is highly biased, which occurs since the power of 5, that relates IRI and damage is very sensible to a few high values. CRFs perform better here for the

same reason that the bias error is smaller, because it is better adjusted to fit high IRI and damage values.

The third result from the study is the observation that socioeconomic data delivers significant regressors. However the prediction of IRI and expected damage does not benefit a great deal, which results in the linear model without socio-economic data being only slightly worse in prediction then the linear model. We expect this to be a side effect of the nature of our dataset. We find very similar socio-economic data in the areas of Sweden. Extrapolation between different countries might be far more dependent on the socioeconomic figures.

7.4 Finland road quality

7.4.1 Available data

The dataset we use for this study is from the Finnish Road administration and is organized in tabular form as depicted in table 7.1. This dataset holds no complex topological road network information (no junctions or crossings) but contains information which segments form seperate roads in the same manner as the swedish dataset. This simplistic topology information could be modeled by linear graphical models as in 2.2.3. Using the more sophisticated general CRF framework yields the same results in this case. The variables are stored for each 100m segment and can be split into discrete and continuous figures.

Table 7.4: Histograms of the discrete figures in the finnish road dataset

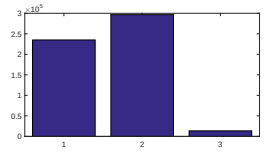
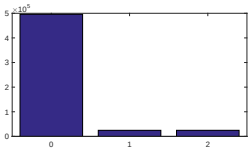
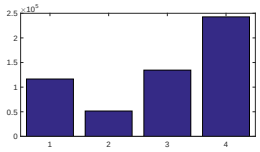
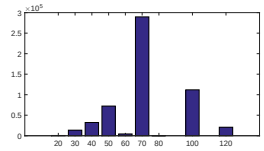
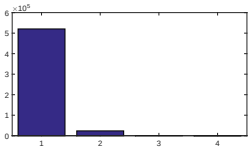
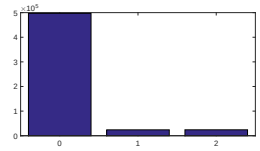
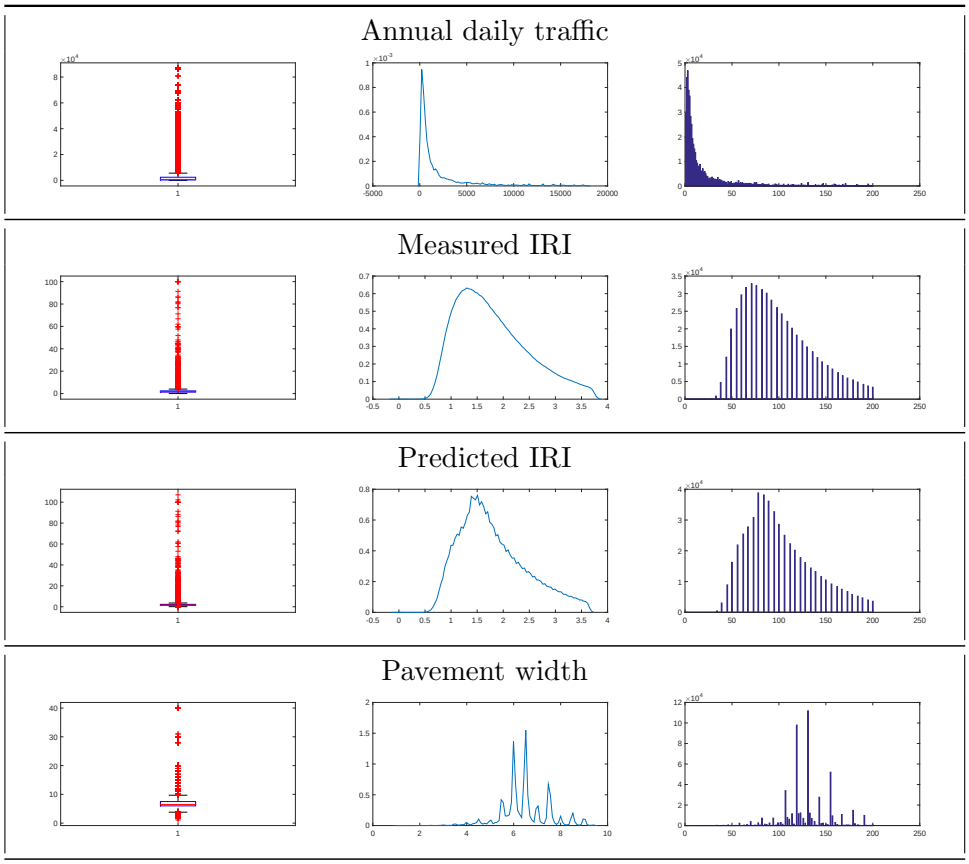
Pavement class {1,2,3}	Direction {1,2,3}	Functional class {1,2,3,4}
		
Speedlimit {30,40,50,60,70,100,120}	Lane {1,2,3,4}	Carriageway {0,1,2}
		

Table 7.5: Boxplots, Gaussian kernel density esitmation and histograms of the continuous figures in the finnish road dataset



7.4.2 Data preparation

To gain a reliable set of IRI measurements we first discard very old measurements and also those which have been made before the last maintenance. Again, we define outliers the standard boxplot way. Outliers are IRI values larger than $q_{.75} + 1.5(q_{.75} - q_{.25})$, values larger than the 75 % quantile plus 1.5 times the interquartile range. The filtered dataset contains 286,000 road segments and is therefore far larger than the swedish road dataset.

We define the road objects with the help of structural breaks in the data. Beginning from the starting point of a road, we aggregate the measure points to a new road segment as long as no regressor changes. Once a regressor changes, we create a starting point for a new road segment and aggregate again as long as all regressors remain constant (with respect to this new starting point). We compute the average IRI for each of these new road segments. This approach stems from the practical perspective because as long as no regressor alters the estimate for the roughness remains constant and always orients itself towards the mean of the roughness. The new dataset contains 27,551 road objects.

7.4.3 Linear Model

Instead of using a standard linear model, we incorporated abrasion by taking into account the period of time which has surpassed since the last maintenance. This is only possible in the datasets which hold the information when a measurement was conducted and when the street was last repaired and the original quality was restored. We denote the difference between the date of measurement and the date of the last maintenance by $\Delta t = t - t_0$. We fit the logarithm of the IRI data to a function of the following kind: We divide the parameter vector $\boldsymbol{\theta}$ into two types: The factors that are not influenced by the maintenance of the road $\{\theta_i | 0 \leq i \leq 7\}$ and a set of parameters which is used for factors that describe the dynamic evolution over time $\{\theta_i | 8 \leq i \leq 10\}$. The results are shown in Table 7.6 and show that every regressor is highly significant and the goodness-of-fit is $R^2 = .36$.

7.4.4 Model comparison

As in the Sweden study, we can also use CRF modeling to obtain a prediction for the IRI or the expected damage. Using the same error measures as in the Sweden study, we get the following results:

7.4.5 Results

We want to emphasize two major results from the study of the Finnish road dataset. First we note that all prediction errors are below the equivalents for the swedish dataset. The bigger dataset and a different set of available regressors explain that difference in performance. Secondly we note that the CRF, that does not use any regressors still performs better than the linear model. This model makes use only of the topological information

Table 7.6: Trained linear model parameter

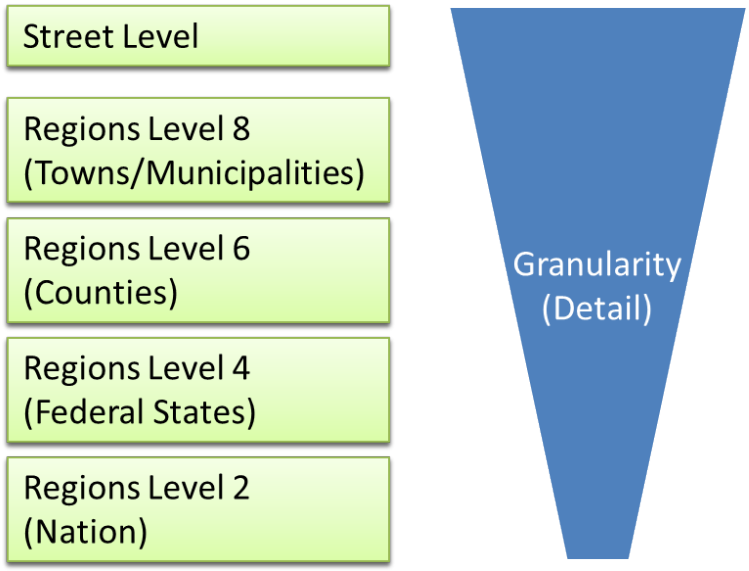
$\boldsymbol{\theta}_i$	Estimate	Observation	p-Value
0	0.8486	constant	0
1	0.0836	Dummy Functional Class	3.03E-66
2	0.0922	Dummy Pavement Class	9.50E-96
3	-0.0775	Dummy Single Carriageway	5.44E-19
4	-0.0505	Dummy Passing Lane	4.29E-09
5	-0.0046	Speed Limit	0
6	-0.0146	Pavement Width	3.63E-21
7	0.0091	ADT^{-1}	1.56E-66
8	0.0383	Log of Maintenance Interval	3.73E-23
9	-0.0031	Maintenance Interval times $ADT^{.2}$	1.13E-05
10	0.0491	Maintenance Interval / Pavement Width	5.15E-35

Table 7.7: Extrapolation results using CRF and linear model

		CRF		Linear Model		Reduced Linear Model	
		RMSE	BIAS	RMSE	BIAS	RMSE	BIAS
Leave 1% out	IRI	0.5	-0.1	0.56	-0.08	0.52	-0.07
	Damage	76.3	-22	103	-29	78	-21.4
Leave 10% out	IRI	0.51	-0.1	0.57	-0.06	0.52	-0.07
	Damage	76	-21	125	-28	77.9	-20.5

of the road network. This shows the high significance of neighboring measurements. This effect is much stronger then in more coarse datasets since the neighbourhood dependency loosens when larger street segments are considered, like we did in the Sweden study. It shows however that small and potent models can be received by roads network topology into consideration.

Figure 7.5: Data organisation diagram for georeferenced data

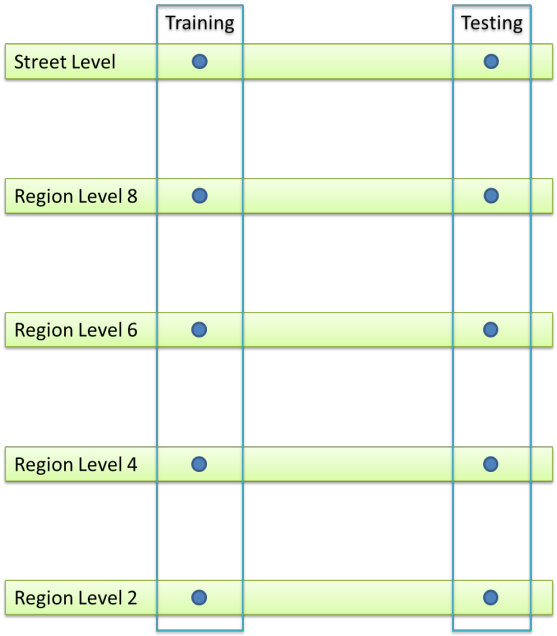


7.5 Extrapolation in the context of VMC

The Fraunhofer VMC (Virtual Measurement Campaign, 1.3) can benefit from using extrapolation techniques in various ways. They can be used for transferring information to areas of interest where essential information is missing. The user intent is usually one of the above (7.1) defined cases. By looking at the data organization of VMC we can define a data transfer schema which accounts for all cases of data transfers including these extrapolation scenarios.

As described in section 1.3 the environment database uses information from OpenStreetmap. Besides street related information there is a lot of socioeconomic data which is not assigned to streets, or portions of a street object, but to areal regions. Almost all information within the VMC environment database is organized in a similar way as in OpenStreetmap. Every piece of information is geolocated. This is true for road related information as well as region related information like demographics or socioenomic figures. In the following schema and in the illustration in Figure 7.6 we will assign the level of granularity in addition to the geolocation to every information set. The geographical location is given or can be tranformed into a set of longitude and latitude coordinates. The level of granularity is related to the administration level within the OSM database. The term 'administration level' refers to the kind of administrative region the information belongs to. Such a region can be a municipality (level 6-8), regions (level 6-4) and countries (level 2). For example the GDP for a federal state in Germany is stored in the

Figure 7.6: Data organisation diagram for georeferenced data

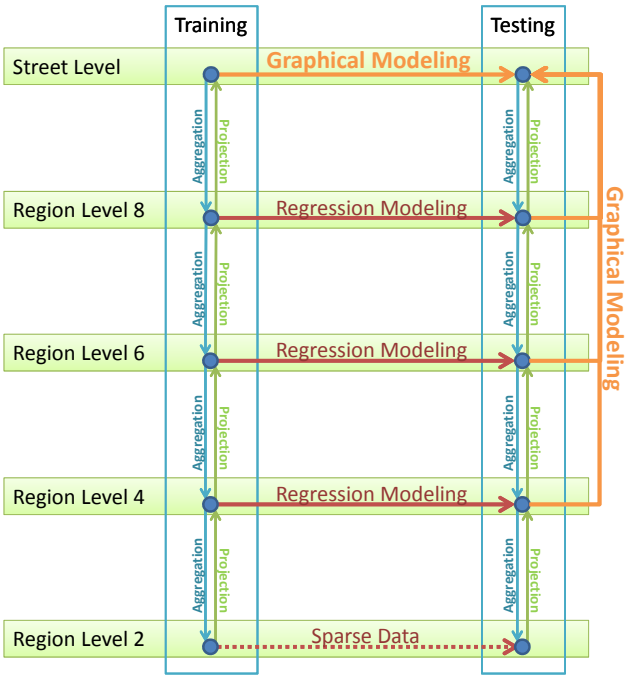


database as the raw figure, together with the longitude and latitude information of the state's border and the administrative level, which is in this case level 4. Depending on a countries administration the interpretations of the administration levels can vary slightly (overview on `http://wiki.openstreetmap.org/wiki/Key:admin_level#admin_level`). In the VMC context the streets are interpreted as additional administration level, which is situated above level 8 in terms of granularity (see Figure 7.5).

7.5.1 Data transfer scheme

Extrapolation as used in this work and in the context of VMC is used for the estimation of information in a target area. Using different inhomogenuous sources of information we also have to consider the level of granularity in such an information transfer. The scheme in Figure shows available data as a blue dots. For the sake of simplicity the figure only shows two regions of interest. Analogously to classification problems, we define the training region as source of information and the testing region as the target area. The different granularity is depicted in vertical direction. Figure 7.7 shows the data transfers that we are using and introducing in the next sections. Such a data transfer is useful whenever the data is scattered through different levels of granularity and needs to be available in a closed form on a specific level. Further the extrapolation of data can also be regarded as data transfer from a set of data in a training region to a target region. The data preparation on training and testing side corresponds to vertical transfers.

Figure 7.7: Data transfer diagram within the VMC database



7.5.2 Vertical data transfer

Changing the granularity of information entities is equivalent to a vertical data transfer in the scheme depicted in figure 7.7. Most commonly information is passed vertically by either aggregation or projection of information. The aggregation of data is used whenever the level of granularity of interest is lower than the available data which is to be transferred. We use a set of aggregation functions that can be used for aggregation, for example sum(), min(), max(), mean() Different figures require different aggregation functions due to their semantics. While a figure like "total road length" or "area" is aggregated by using the sum, the "GDP" or "IRI" is aggregated usind the mean. For accurate results it is however necessary to use weighted means or even more complex aggregation functions. Projection describes the process of copying a value of a region of low granularity to all regions of higher granularity which geographically can be located within the more coarse region. In the Swedish dataset used in the study above it is possible to project all socioeconomic figures down to the individual street segments. This yields a rather low entropy distribution but without assuming some distribution for the higher granularity levels it is the only option of an upwards data transfer. The data preparation we described in the above studies can be regarded as the vertical transfer of data (by projection) within the training dataset.

7.5.3 Horizontal transfer

The horizontal transfer is the data transfer between a region and another. The scenarios for extrapolation in 7.1 can all be viewed as an horizontal data transfer on street level. Therefore the 'Graphical modeling' between the training and testing dataset denotes the extrapolation procedure used in the studies above. Extrapolation on other granularity levels than street level cannot easily be achieved by using graphical models. This is due to two reasons:

1. The topological structure is not uniquely defined and artificially designed topologies are usually of poor importance to the model.

2. The amount of entities (regions) is usually small compared to the number of road objects. This makes the classification of continuous data (7.2.2) loose too much information.

To transfer data in other levels than the street level, we use linear modeling. This is done the exact same way we used linear models for extrapolation on the street level in the above studies.

7.5.4 Mixed transfers

One way to improve the data distribution on the levels of higher granularity is to use graphical models whenever data is transferred to the street level. In Figure 7.7 this procedure is depicted as the rightmost set of arrows, annotated by 'Graphical modeling'. A graphical model is used in order to use the topology information, which is not explicitly depicted and only available (or at least uniquely interpretable) on street level. This method can be used if the training dataset has the target figure available on the street level as well. Than the procedure can be summarized as follows:

1. Project the figure of interest to the street level, on the training and testing side

2. On the training side there the target figure is now available in the high granularity, per segment form of high entropy and the low entropy projection information.

3. Train a graphical model with just the topological information and the projected figure with the high granularity target figure as classification target.

4. Use the trained model to achieve a high entropy target value distribution in the testing region

This procedure can for example be used to obtain a realistic IRI distribution for the 100m segments in the testing region by only knowing an average IRI in each municipality. This kind of data transfer can be regarded as a vertical transfer (IRI from a region to street level) by using a model trained like we do in the horizontal transfer (by using the information in the training dataset).

8 Conclusion

8.1 Result

8.1.1 Conclusions regarding Optimization

The target of this part of our work is to find an optimization method which can deliver good CRF training characteristic when using standard desktop computer hardware. Since the models we want to train can be of huge magnitudes in terms of size and complexity it is often not possible to rely on the well established non-linear algorithms such as LBFGS.

We analyzed a set of stochastic optimization techniques. By adjusting the Stochastic Average Gradient approach from Roux et al. [RSB12], we found a way to maintain convergence behaviour for fixed step sizes far greater than the analytic boundary ($\delta \approx 0.5 \gg 0.03125$).This enables us to optimize Conditional Random Fields with many nodes ($n > 10^5$) and with big parameter spaces ($\dim \boldsymbol{\theta} > 10^3$) efficiently.

Due to the lack of available data sets of such proportions we developed a way to construct artificial street graphs. Those graphs were parametrized by statistics from real world street data of finland and sweden. By researching the different parametrizations for connectivity, graph size and feature weight distributions showed that the convergence behaviour is mainly dependent on the graph size n.

We conclude in congruency with the suggestions of Roux et al. [RSB12], that whenever it is computationally suitable, one should use standard non-linear optimization and in particular LBFGS to train Conditional Random Fields. Whenever complete LBFGS iterations are too expensive it is highly advisable to use stochastic approaches. Within the group of first order optimization techniques we found the 'Robust Locally Weighted Stochastic Average Gradient' to be the method of choice, for it is independent of the parameter space and allows high constant step sizes.

8.1.2 Conclusions for Extrapolation

Using Conditinal Random Fields as a model class allowed us to use markovian data dependencies as well as local features in a similar manner as linear models. In a study on the Finnish and Swedish road network data, we showed that even a very small markovian model (only edges) is as powerful as a linear model with a wide variety of regressors. A further benefit you gain from using graphical models is that inference can deliver good

estimates of the target quantities on whole subsets of the graph. In order to simulate interconnected structures, for example a complete route within the road network, this yields far more realistic samples than a point estimator by a linear model.

8.2 Related Work

We mentioned related work throughout this work along with the different topics. At this point we want to point out a set of related work concerned with the training of graphical models and stochastic optimization.

8.2.1 Boosting

Boosting describes a set of algorithms that are used for training models by using a combination of weak learners to form one strong learner. In the case of CRF such a weak learner can directly be associated with a feature function. A good feature selection algorithm can dramatically improve the performance when training a graphical model. Therefore boosting can be viewed as an alternative or an addition to the numerically motivated feature selection that we use by using penalty terms. There are several implementations and advances in the field, which are often shown to perform well on linear graph problems and tree graphs. Especially approaches like Boosted random fields ([TMF04]) and Virtual Evidence Boosting ([LCFK07]) have been taken into consideration for our model. The field of boosting is very wide, delivering special boost variants for different model assumptions. We have not discovered an already established Boosting approach for Conditional Random Fields on general graphs that works on huge graphs. However reducing the CRF to local linear or tree like graphs can enable algorithms like the algorithms mentioned above, AdaBoost or Gradient Tree Boosting [DAB04] to perform efficient model training by feature selection.

8.2.2 Higher Order Optimization

Using Semi-Newton algorithms for optimization is usually computationally to expensive for standard hardware when dealing with huge problems. The group around Nic Schraudolph at NICTA, Australia published a series of papers that show how to use stochastic approaches with non fixed step sizes, using higher order information.

Online optimization by Nicta [STVS09]

The paper of Sunehag, Trumpf, Vishwanathan and Schraudolph summarizes Schraudolphs work. They give the very basic definition of online optimization as

$$C_t(\boldsymbol{\theta},) = \frac{1}{t}\sum C(\boldsymbol{\theta}, z_i)$$

with C_t being an approx to the real objective $C(\boldsymbol{\theta}) = E_z C(\boldsymbol{\theta}, z)$.
Further they introduce gradient based approaches with predefined metric κ. This metric influences the gradient. The most general notion of a stochastic gradient descent step is therefore

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - a_t B_t \nabla C(\boldsymbol{\theta}_t, z_t)$$

Where B_t is the transformation of the standard metric, a_t be the stepsize.
The paper gives examples where stochastic optimization is needed and what an objective function might look like. Classification via loss + regularization function, filtering ($C_t = \sum (y_i - wx_t)^2 \rightarrow V_t = C_t^{-1}$)
The next example introduces the concept of expert advices (series of loss functions L_t) with the goal of having a small accumulated loss. In the example they also discuss the possibility of using reparametrization of the loss function in order to obtain a loss that satisfies convergence criteria.
The authors state the multivariate Robbins Monroe procedure of Blum [Blu54] and adapt the work of Bottou and LeCun [BL05] for the use of variable metrics. They show that $\boldsymbol{\theta} \rightarrow \boldsymbol{\theta}^*$ using a set of conditions and the Robbins-Siegmund theorem. Under a further condition, strong convexity near optimum, convergence speed can be shown to be $O(n^{-1})$ (referencing Bottou and LeCun [BL05])
The online version of LBFGS by Schraudoplh can now be viewed as variable metric. Schraudolph already used a trust region parameter λ to assure that the eigenvalues of the update metric are bound (low and high). Therefore the theorem of this paper can be used to proof the convergence speed of Schraudolphs oLBFGS.
The paper further applies the theorem to expert advice systems and concludes.

Stochastic Meta Descent

One major problem in stochastic gradient approaches is to find a suitable stepsize strategy. The stochastic meta descent (SMD, [Sch99]) provides a way to adapt the gain for the current optimization step. This is achieved by using a vector gain $\boldsymbol{\eta}_t$ instead of a one-dimensional step size:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \boldsymbol{\eta}_t \cdot \nabla f(\boldsymbol{\theta}_t)$$

We look at each iteration step as an optimization problem in $ln\boldsymbol{\eta}$, regarding the gradient as function of $\boldsymbol{\theta}(\boldsymbol{\eta})$

$$\begin{aligned} \ln \boldsymbol{\eta}_{t+1} &= \ln \boldsymbol{\eta}_t - \mu \nabla_{\ln \boldsymbol{\eta}} f(\boldsymbol{\theta}(\boldsymbol{\eta})) \\ \boldsymbol{\eta}_{t+1} &= \boldsymbol{\eta}_t \cdot \exp\left(\mu \nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}) \cdot \nabla_{\ln \boldsymbol{\eta}} \boldsymbol{\theta}(\boldsymbol{\eta})\right) \end{aligned}$$

Schraudolph suggests to approximate the exponential function in order to be robust against outliers and to increase performance. He suggests the following iterative process

$$\begin{aligned} \boldsymbol{\eta}_{t+1} &= \boldsymbol{\eta}_t \cdot \max\left(\frac{1}{2}, 1 - \mu \nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}) \cdot \mathbf{v}_t\right) \\ \mathbf{v}_{t+1} &= \lambda \mathbf{v}_t - \boldsymbol{\eta}_t \cdot (\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}) + \lambda \mathbf{H}_t \mathbf{v}_t) \end{aligned}$$

The parameters μ and λ are free parameters that have to be tuned. Although H_t as the Hessian is usually of high complexity Schraudolph shows [Sch02] that calculating $\mathbf{H}_t\mathbf{v}_t$ can also be done efficiently.

BFGS on stochastic objective functions

Schraudoph et al. [SYG07] describe a way to extend BFGS and LBFGS to stochastic objective functions. Since these variants can very well be applied to online problems we will use their notation and call these variants oBFGS and oLBFGS. The two main difficulties when using (L)BFGS with stochastic objective functions are:

- Stepsize conditions do not guarantee global convergence
- With every iteration the objective function changes and not only the point in parameter space $\boldsymbol{\theta}_k$. This makes finite differences in the gradient be inaccurate approximations for the Hessian updates.

The stepsize problem can be resolved by using a predefined decay strategy. We discuss valid strategies in 5.3.2.
The problem of invalid finite differences can be solved by evaluating them on the same sub problem. Let f_k denote the objective function of the subproblem in iteration k. Then we calculate

$$\mathbf{y}_t = \nabla f_t(\boldsymbol{\theta}_{t+1}) - \nabla f_t(\boldsymbol{\theta}_t)$$

This differs from the straighforward application of the BFGS algorithm since the first term would not be calculated in any iteration. It doubles the number of gradient evaluations but Schraudolph et al. [SYG07] show that this procedure performance can be a vast improvement over standard BFGS in the context of expensive function evaluations.

Algorithm 8.1 oBFGS Method

1: t := 0
2: $\mathbf{B}_0 = \epsilon\mathbf{I}$
3: **while** not converged **do**
4: (a) $\mathbf{p}_t = -\mathbf{B}_t\nabla g(\boldsymbol{\theta}_t)$
5: (b') $\eta_t = \frac{1}{c}\text{decay}(t)$
6: (c) $\mathbf{s}_t = \eta_t\mathbf{p}_t$
7: (d) $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \mathbf{s}_t$
8: (e') $\mathbf{y}_t = \nabla f_t(\boldsymbol{\theta}_{t+1}) - \nabla f_t(\boldsymbol{\theta}_t)$
9: (f) if $t = 0 : \mathbf{B}_t := \frac{\mathbf{s}_t'\mathbf{y}_t}{\mathbf{y}_t'\mathbf{y}_t}\mathbf{I}$
10: (g) $\rho_t = (\mathbf{s}_t'\mathbf{y}_t)^{-1}$
11: (h') $\mathbf{B}_{t+1} = (\mathbf{I} - \rho_t\mathbf{s}_t\mathbf{y}_t')\mathbf{B}_t(\mathbf{I} - \rho_t\mathbf{y}_t\mathbf{s}_t') + c\dot{\rho}_t\mathbf{s}_t\mathbf{s}_t' +$
12: (i) $t := t + 1$
13: **end while**
14: return $\boldsymbol{\theta}_t$

Algorithm 8.2 online L-BFGS Method

t := 0
$\mathbf{B}_0 = \epsilon\mathbf{I}$
while not converged **do**
    (a1) $\mathbf{p}_t = -\nabla g(\boldsymbol{\theta}_t)$
    for $i := 1, 2, \ldots, \min(t, m)$ **do**
        (a2) $\alpha_i = \frac{\mathbf{s}'_{t-i}\mathbf{p}_t}{\mathbf{s}'_{t-i}\mathbf{y}_{t-i}}$
        (a3) $\mathbf{p}_t := p_t - \alpha_i \mathbf{y}_{t-i}$
    end for
    (a4) if $t > 0 : \mathbf{p}_t := \frac{\mathbf{s}'_{t-1}\mathbf{y}_{t-1}}{\mathbf{y}'_{t-1}\mathbf{y}_{t-1}}\mathbf{p}_t$
    for $i := \min(t, m), \ldots, 2, 1$ **do**
        (a5) $\beta = \frac{\mathbf{y}'_{t-i}\mathbf{p}_t}{\mathbf{s}'_{t-i}\mathbf{y}_{t-i}}$
        (a6) $\mathbf{p}_t := \mathbf{p}_t + (\alpha_i - \beta)\mathbf{s}_{t-i}$
    end for
    (b') $\eta_t = \frac{1}{c}\text{decay}(t)$
    (c) $\mathbf{s}_t = \eta_t \mathbf{p}_t$
    (d) $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \mathbf{s}_t$
    (e') $\mathbf{y}_t = \nabla f_t(\boldsymbol{\theta}_{t+1}) - \nabla f_t(\boldsymbol{\theta}_t)$
    (i) $t := t + 1$
end while
return $\boldsymbol{\theta}_t$

8.3 Outlook

Optimization
There are several other approaches to nonlinear optimization which have not all been evaluated by us. Different stochastic variants of Conjugate Gradient approaches might be of great potential although initial tests in our simulation environment were not too promising.
The robust kernel weighted SAG variants that we proposed are one of many possible ways to enhance standard SAG. There are still various other ways to make enhancements that have not been (fully) evaluated in our work. An incomplete list of possible SAG enhancements:

- Using an optimized weighting function:
  The weighting function, as it is used in section 5.4.3 could be defined as a function of the iteration i, the current position in parameter space $\boldsymbol{\theta}$ and an additional weighting parameter $\alpha \in \mathbb{R}^n$ and be optimized with regard to α on a training data set. The same could be done with the kernel based techniques.

- Using adaptive step sizes:
  There are uncounted possibilities to implement adaptive step sizes in stochastic optimization algorithms. We implemented and worked with many different approaches that did not deliver stable results for all possible set ups while maintaining good convergence properties. However we strongly suggest that a combination of Stochastic Average Gradient and the step size adaption in the Stochastic Meta Descent algorithm has high potential.

- Analytical boundaries:
  We strongly belief that the analytical boundaries for step sizes when using our SAG variants are the exact same as for the standard SAG (by Roux et al [RSB12]). There might be potential for better boundaries when using even other variants of the SAG.

- Meta algorithmic:
  The combination of multiple stochastic optimization runs is also possible. We propose to use SAG with high step sizes and monitor convergence and to restart with lower step sizes if the monitoring indicates divergence. The parallel execution of multiple optimizers and a simulated-annealing type meta algorithmic could yield a powerful alternative.

- Technical aspects:
  We used multi-threaded optimization in our simulations. However optimizing the algorithms for parallel executions and developing GPGPU optimized algorithms might further increase the performance. Using pseudo likelihood methods is a great foundation for parallel solvers.

Modeling
Regarding the model itself, we think that CRF has a very high potential and is a great

and powerful model to deal with street graph problems. Using a multi-templated version of the CRF can simplify the problem in the following way. One template is used for the very common situation of two road segments that are situated next to each other. In model terms, all nodes with cardinality two. For all other situation we would use different templates. This makes the complete CRF disconnect into a set of separate linear graphs during training. For inner city situations this is very well suited but for rural areas with low street connectivity this should be a valid and especially efficient modeling approach.

Real World Data

At some point during this work the need for an artificial street set up became inevitable. This is mainly due to the fact that the preparation of a consistent dataset containing road roughness data and further information is a very time consuming and difficult task. With the every growing databases in the context of open data, such a dataset will hopefully be available in the near future. Using stochastic optimization for conditional random fields can then be used on real world data to allow for good extrapolation of street data.

9 Appendix

In this appendix we provide the analysis results of all simulation runs.

9.1 Artificial Topology

With the following simulations we are looking for parameters of the artificial topologies, that have significant impact on the optimization process and results. The impact on the optimization process is monitored by looking at the convergence behavior of a standard L-BFGS implementation 5.2.1.
The impact on the resulting model is evaluated by looking at a histogram of grouped feature function weights. We group the features as follows:

- Edge: Edge-based features. In the simulation runs we used an Edge-based delta function and an exponential distance function
- VI(i): Vertex-indicator function with i dimensions.

The variable setup for the simulation uses only discrete observable variables. Therefore all vertex-based features are vertex-indicator functions.

Table 9.1: Simulation parameter setup

Simulated graph	
Topology setup	$\{T1\text{-}T6\}$
Number of vertices	$\{1000, 10000, 100000\}$
Street length distribution (σ)	{1.0,5.0}
Weight distributions	$W1$
Optimization	
Objective function	$O0$
Algorithm	L-BFGS,SGD
Linesearch	L-BFGS with Wolfe-Backtracking
Start-values	$N(0, 1)$

Distribution of feature weights (dependent on graph size n)

Figure 9.1: Topology: T1, $\sigma = 1.0$

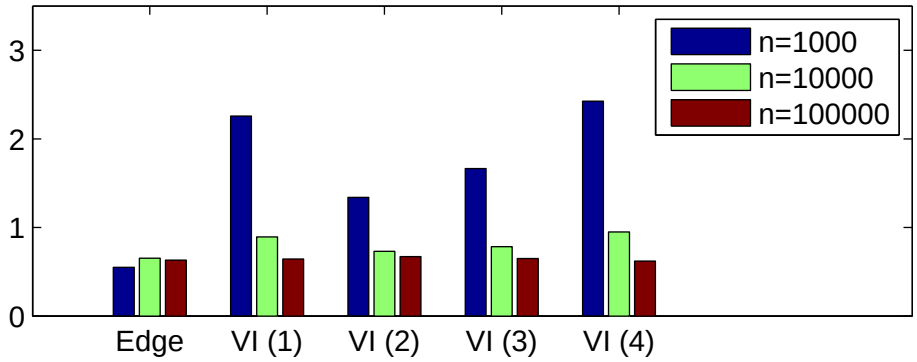


Figure 9.2: Topology: T2, $\sigma = 1.0$

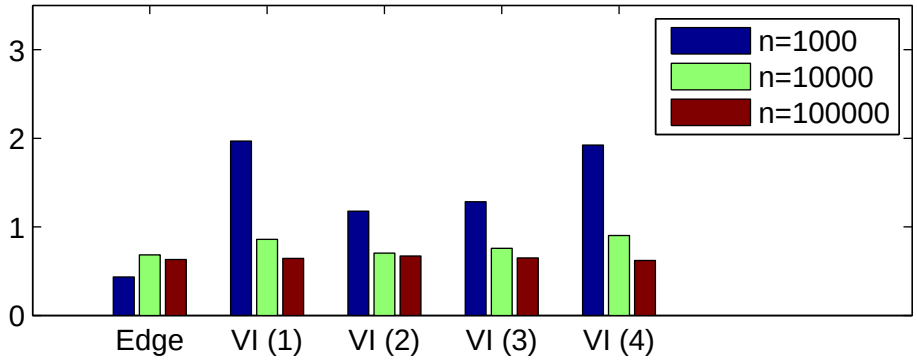


Figure 9.3: Topology: T3, $\sigma = 1.0$

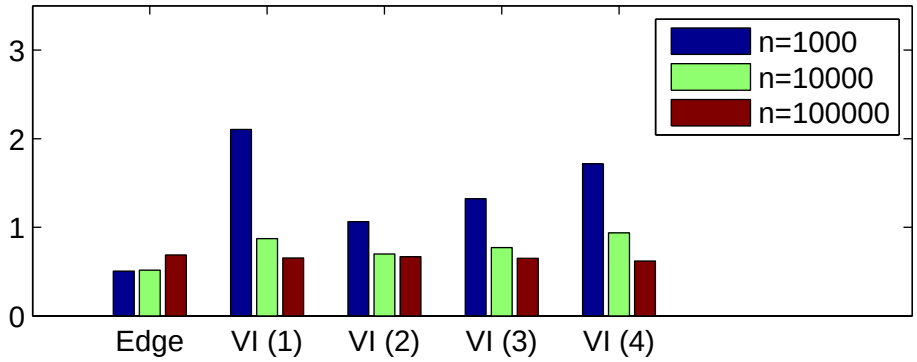


Figure 9.4: Topology: T4, $\sigma = 1.0$

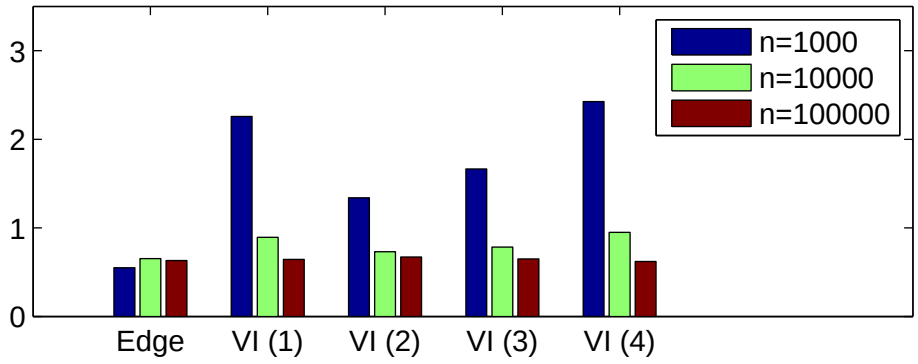


Figure 9.5: Topology: T5, $\sigma = 1.0$

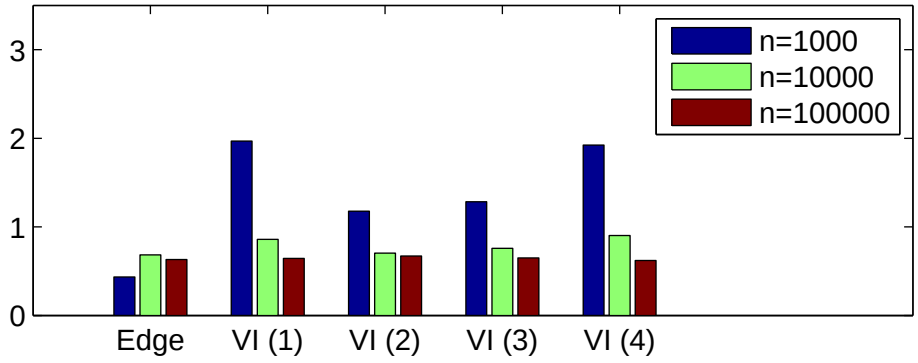
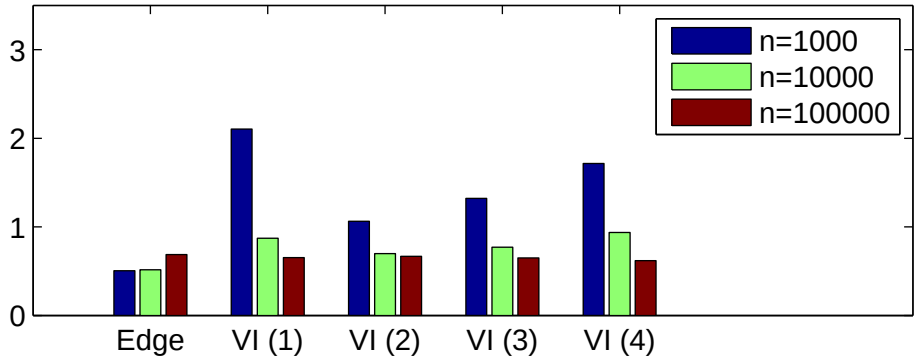


Figure 9.6: Topology: T6, $\sigma = 1.0$



Convergence of LBFGS (dependent on graph size n)

Figure 9.7: Topology: T1, $\sigma = 1.0$

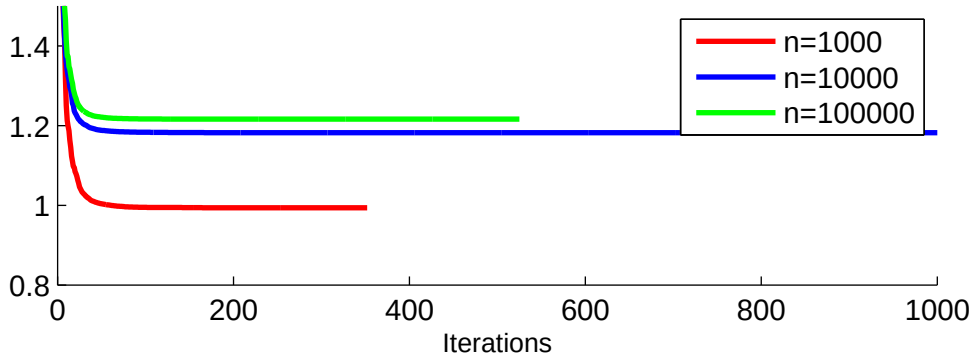


Figure 9.8: Topology: T2, $\sigma = 1.0$

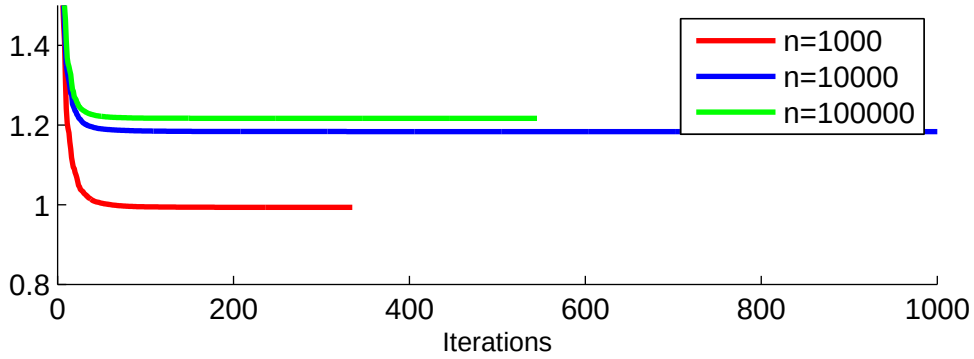


Figure 9.9: Topology: T3, $\sigma = 1.0$

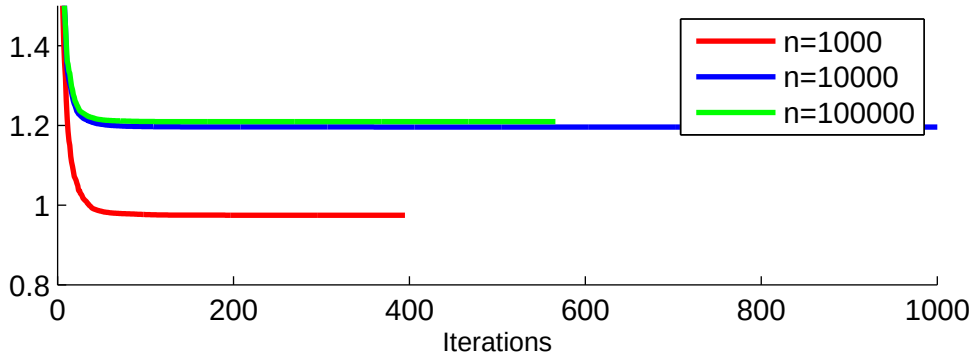


Figure 9.10: Topology: T4, $\sigma = 1.0$

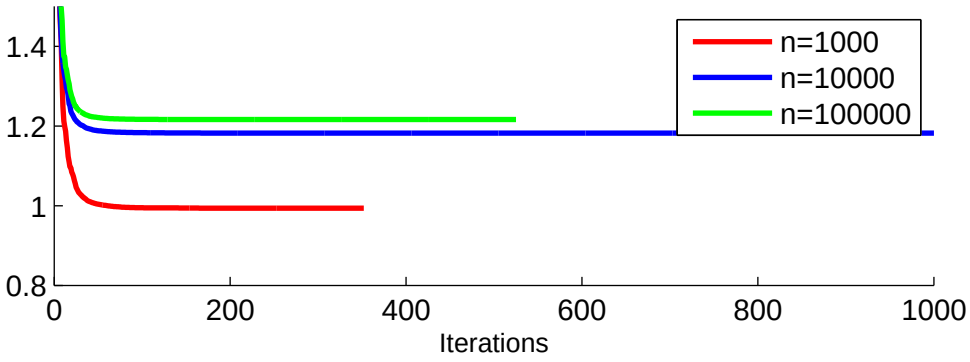


Figure 9.11: Topology: T5, $\sigma = 1.0$

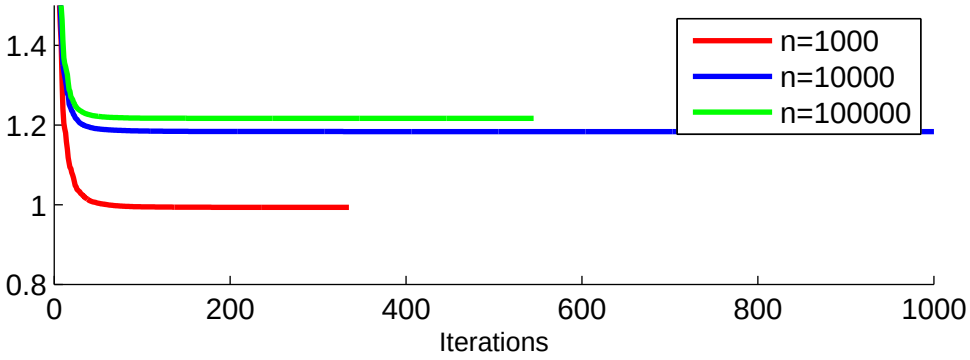
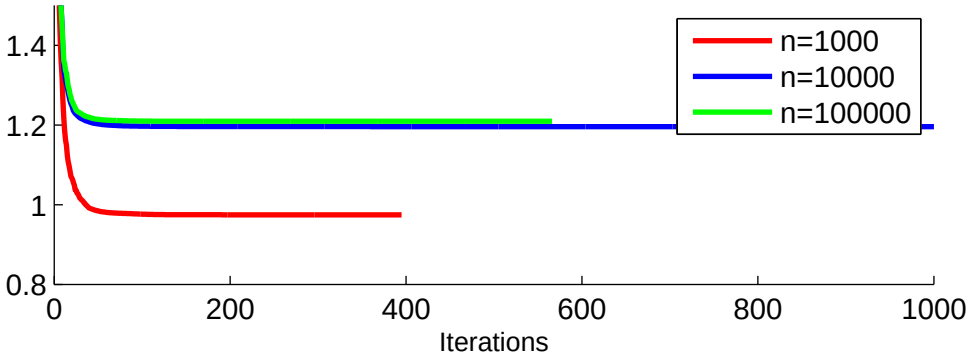


Figure 9.12: Topology: T6, $\sigma = 1.0$



Distribution of feature weights (dependent on street length variance σ)

Figure 9.13: Topology: T1, $n = 1000$

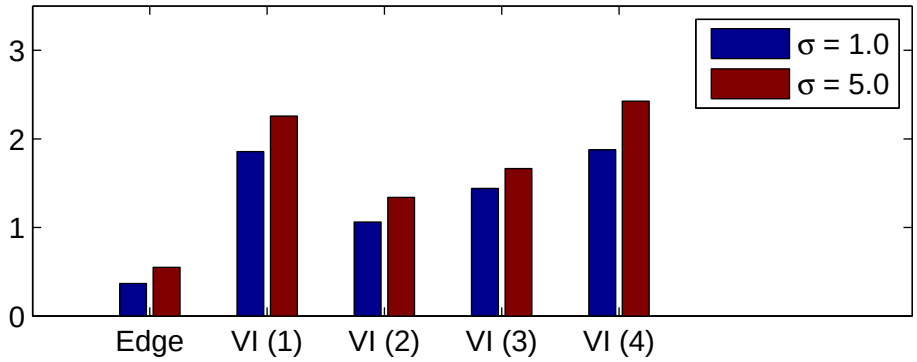


Figure 9.14: Topology: T2, $n = 1000$

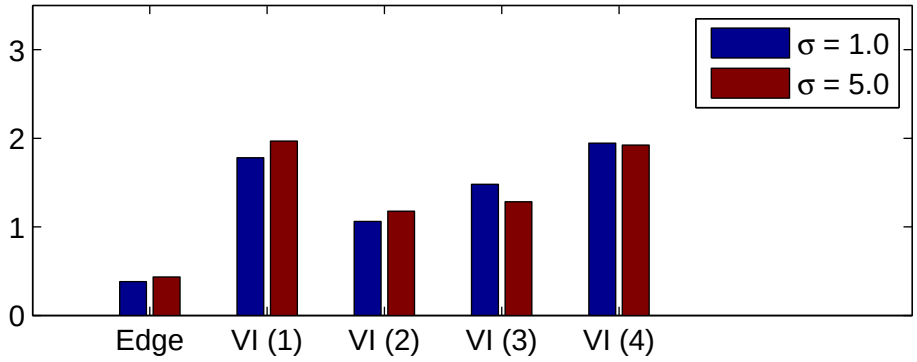


Figure 9.15: Topology: T3, $n = 1000$

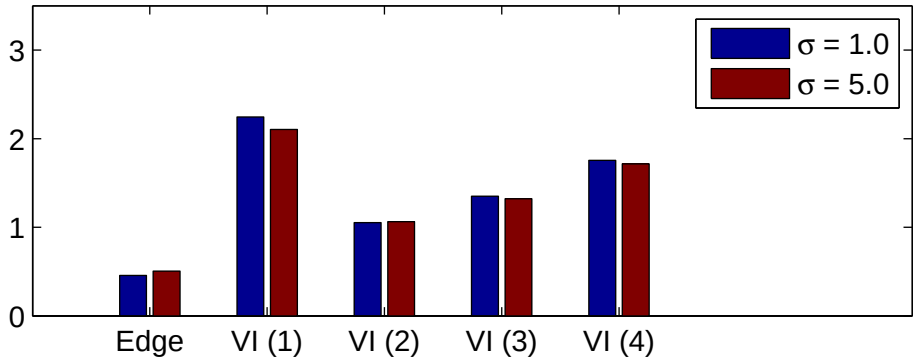


Figure 9.16: Topology: T4, $n = 1000$

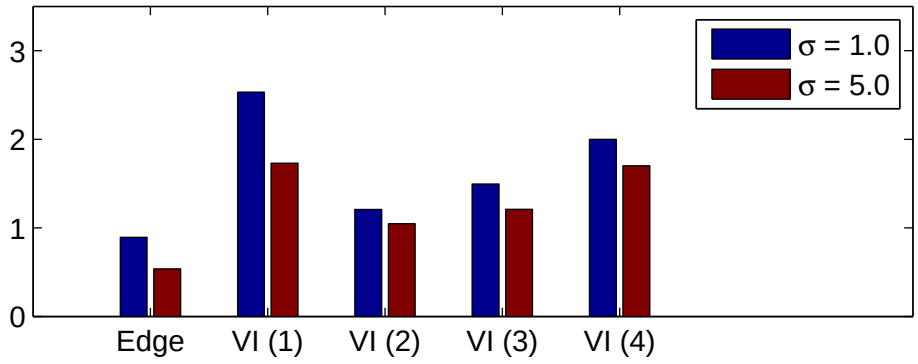


Figure 9.17: Topology: T5, $n = 1000$

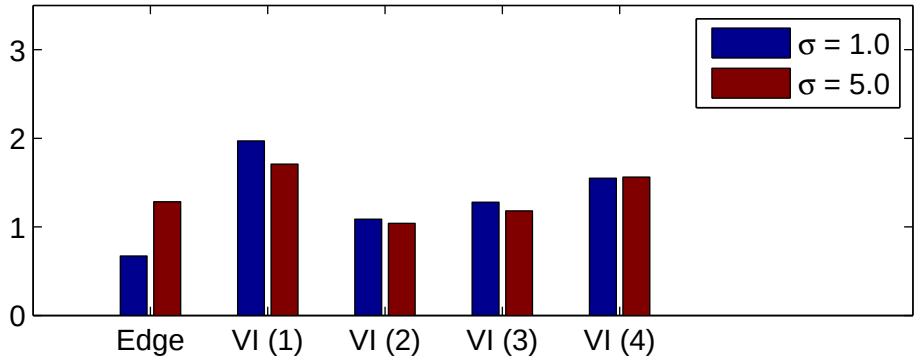


Figure 9.18: Topology: T6, $n = 1000$

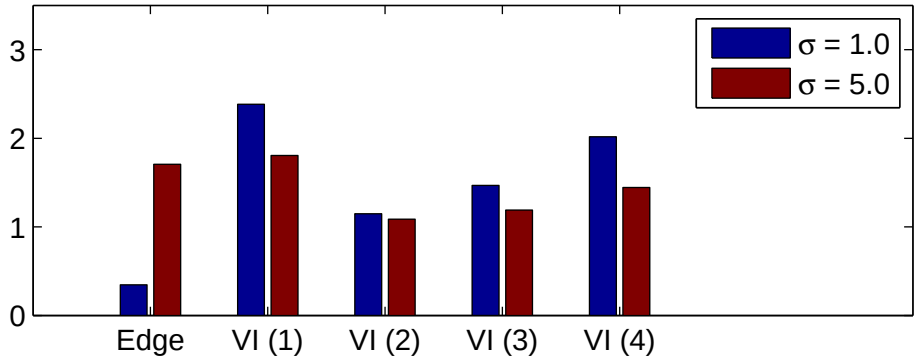


Figure 9.19: Topology: T1, $n = 10000$

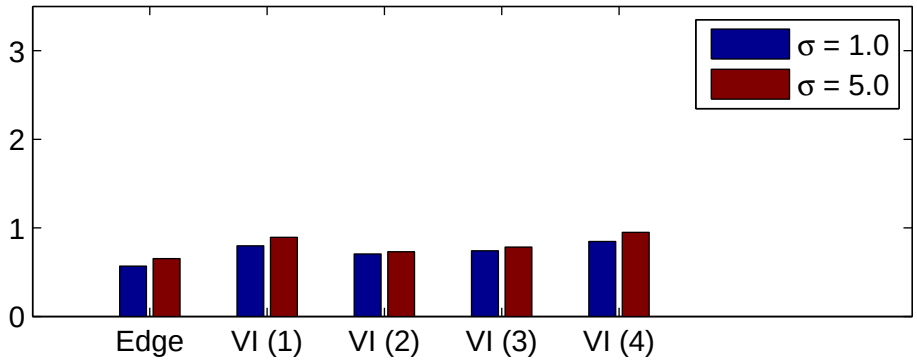


Figure 9.20: Topology: T2, $n = 10000$

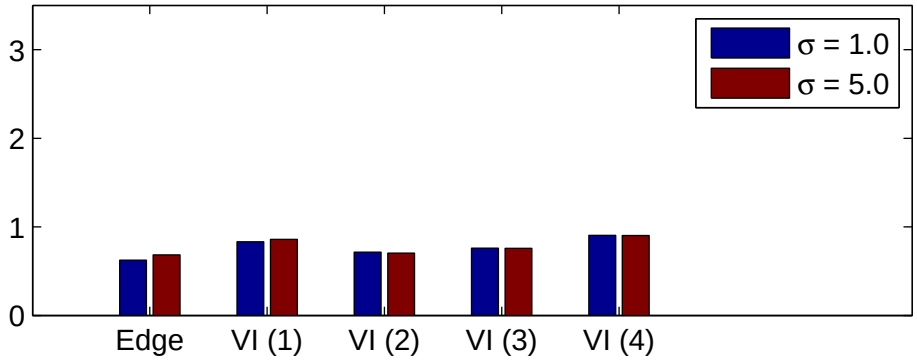


Figure 9.21: Topology: T3, $n = 10000$

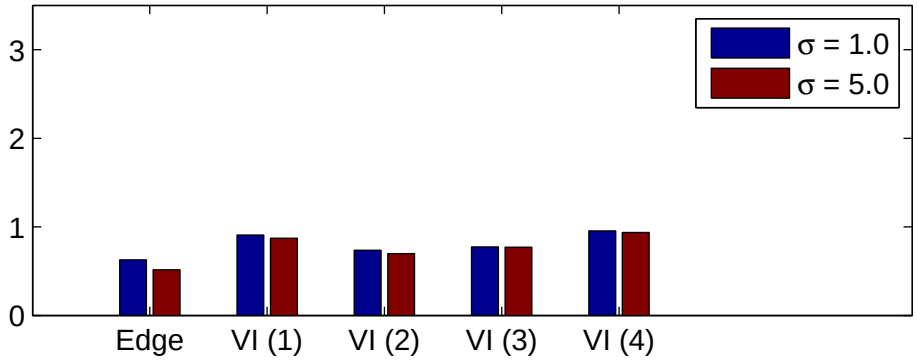


Figure 9.22: Topology: T4, $n = 10000$

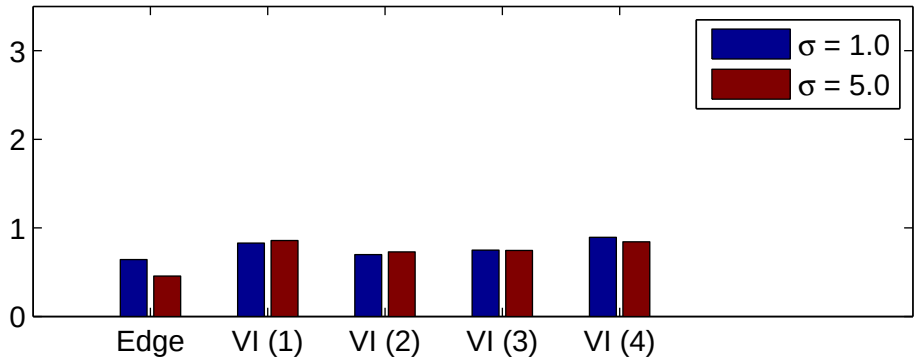


Figure 9.23: Topology: T5, $n = 10000$

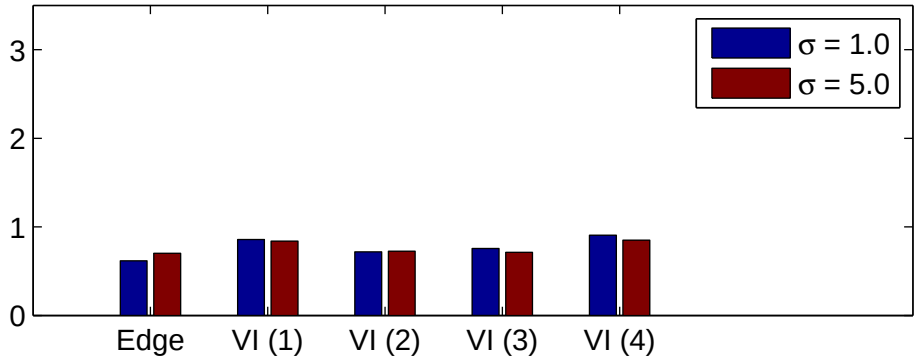


Figure 9.24: Topology: T6, $n = 10000$

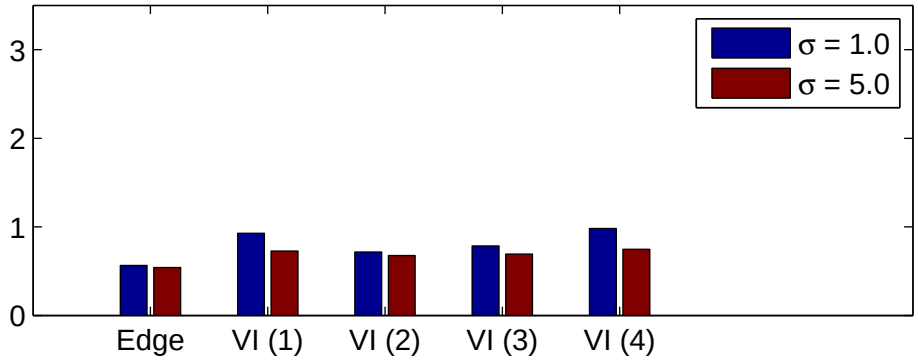


Figure 9.25: Topology: T1, $n = 100000$

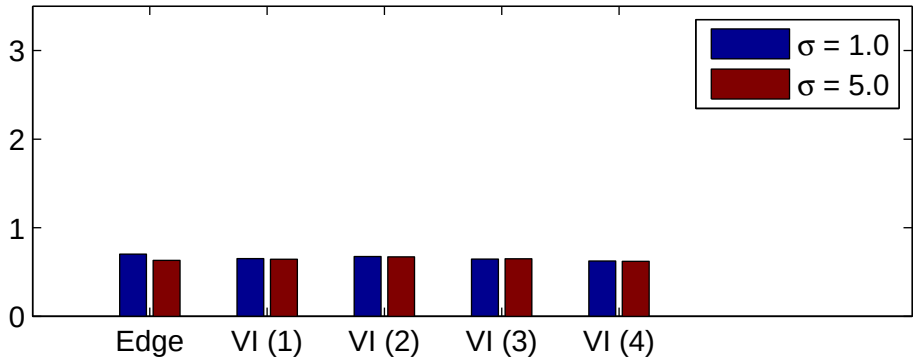


Figure 9.26: Topology: T2, $n = 100000$

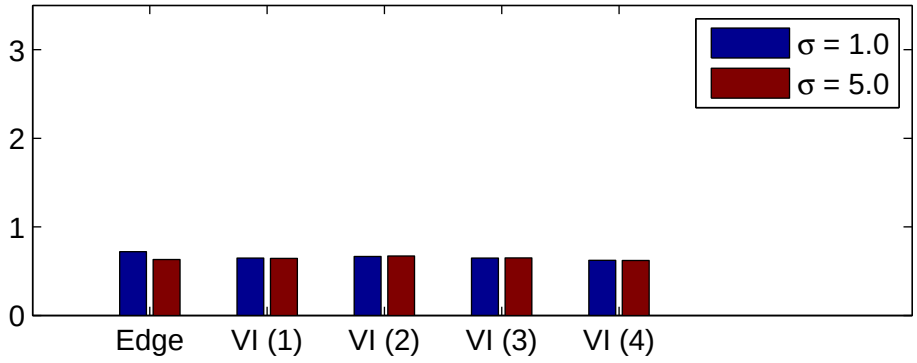


Figure 9.27: Topology: T3, $n = 100000$

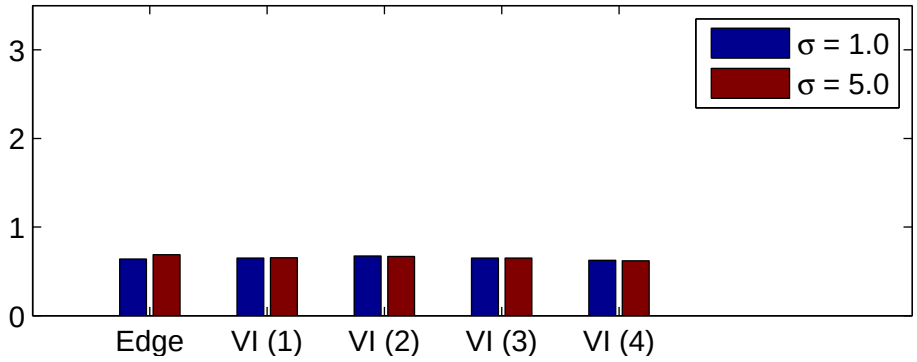


Figure 9.28: Topology: T4, $n = 100000$

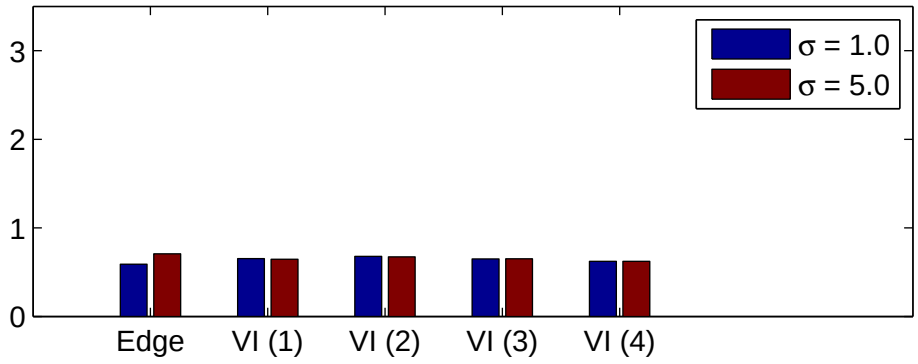


Figure 9.29: Topology: T5, $n = 100000$

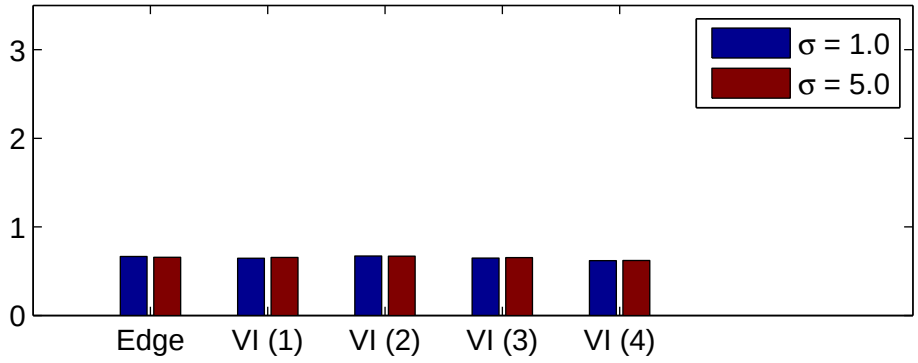
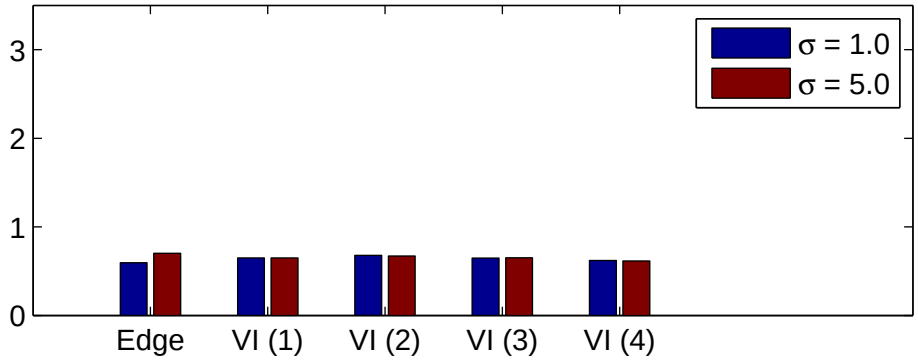


Figure 9.30: Topology: T6, $n = 100000$



Convergence of LBFGS (dependent on street length variance σ)

Figure 9.31: Topology: T1, $n = 1000$

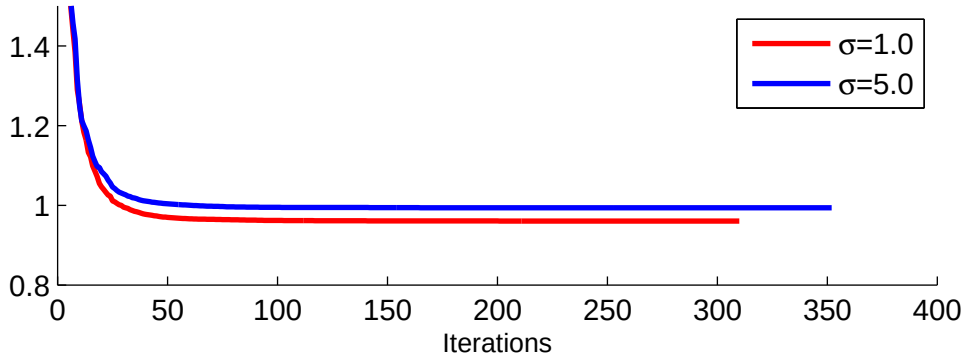


Figure 9.32: Topology: T2, $n = 1000$

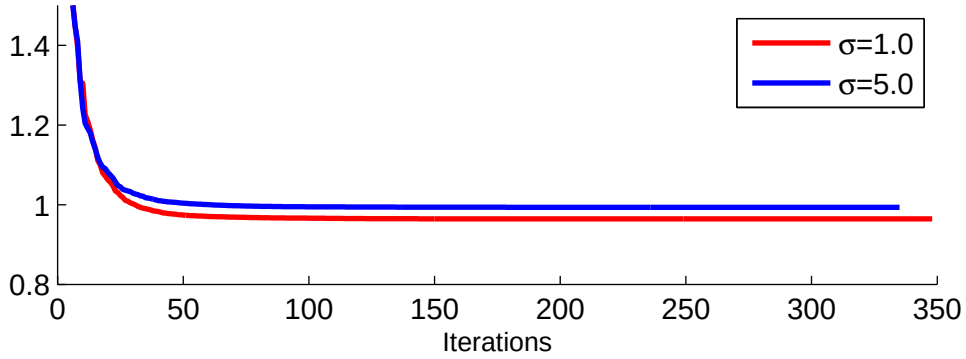


Figure 9.33: Topology: T3, $n = 1000$

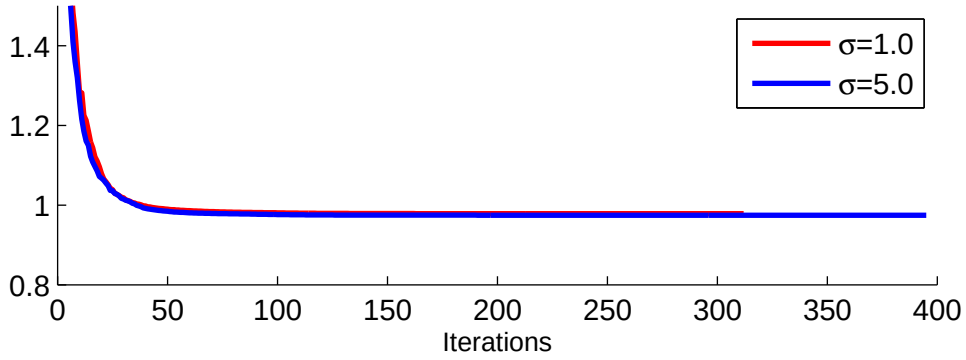


Figure 9.34: Topology: T4, $n = 1000$

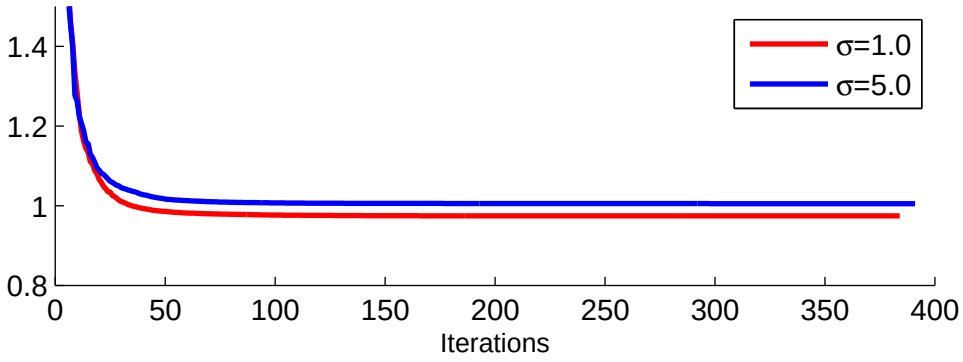


Figure 9.35: Topology: T5, $n = 1000$

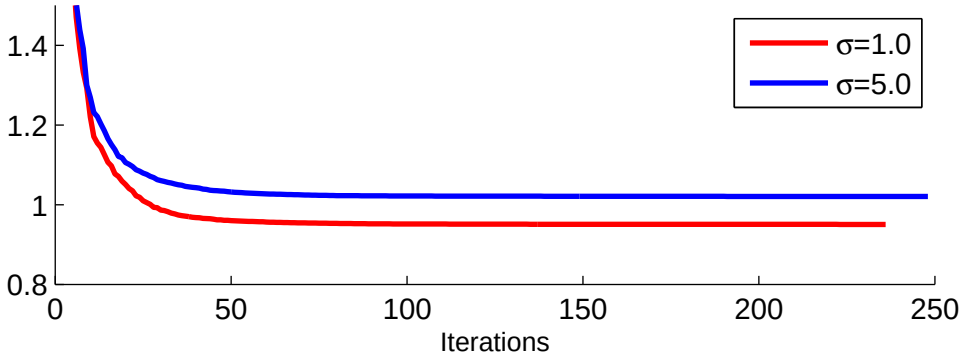


Figure 9.36: Topology: T6, $n = 1000$

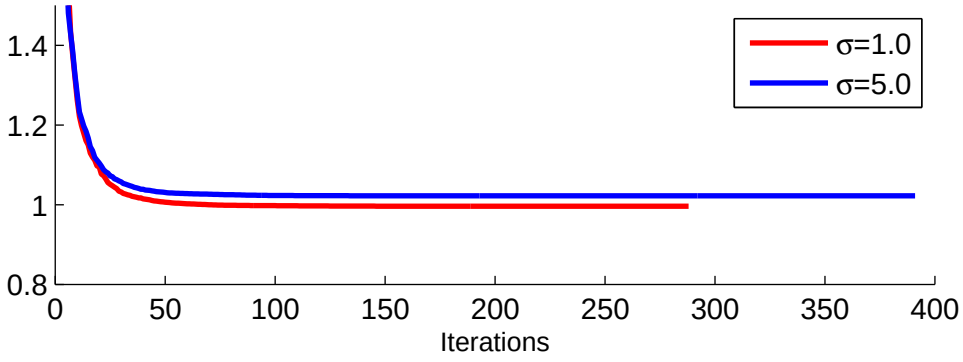


Figure 9.37: Topology: T1, $n = 10000$

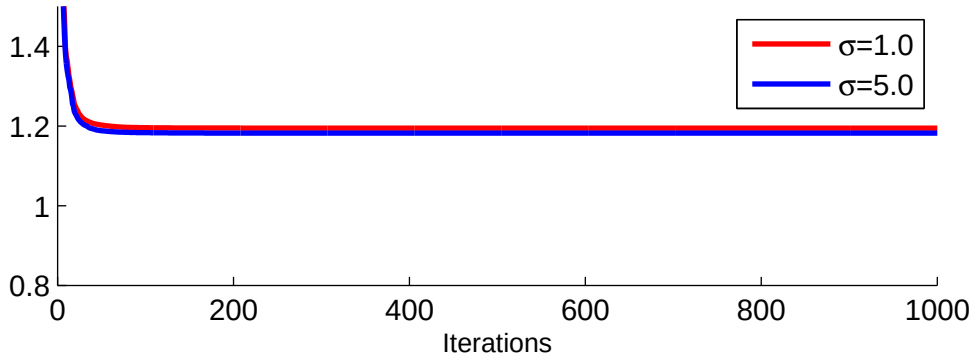


Figure 9.38: Topology: T2, $n = 10000$

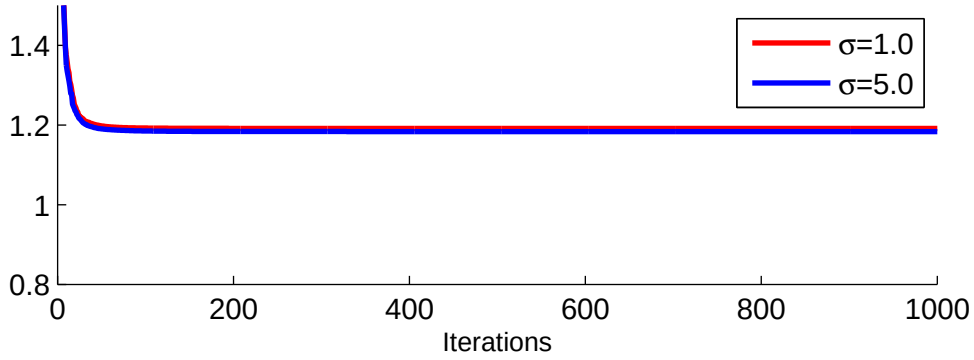


Figure 9.39: Topology: T3, $n = 10000$

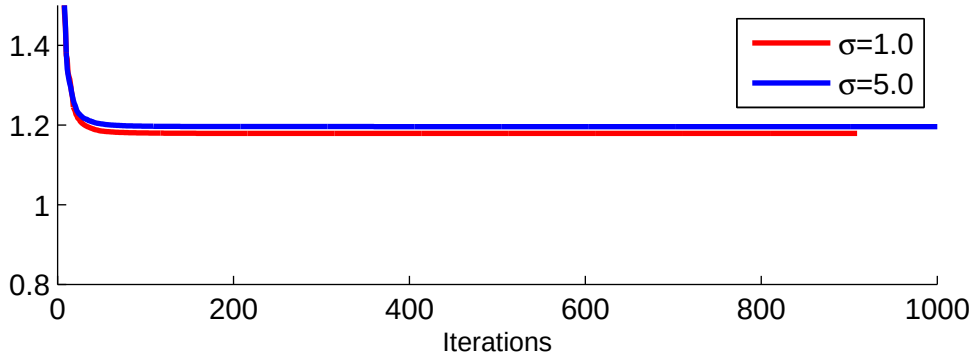


Figure 9.40: Topology: T4, $n = 10000$

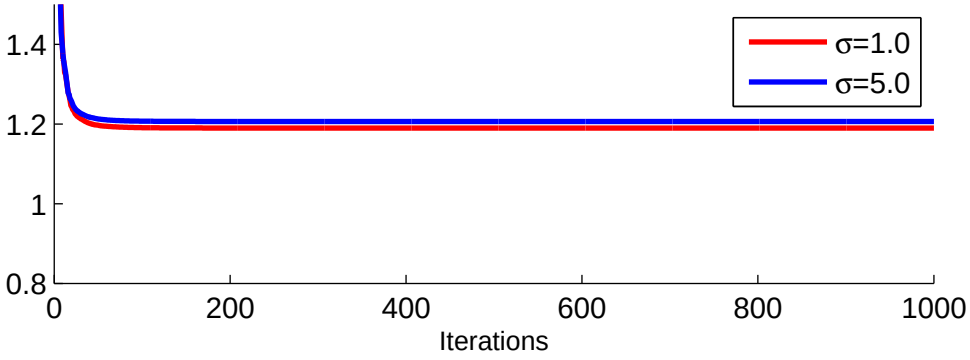


Figure 9.41: Topology: T5, $n = 10000$

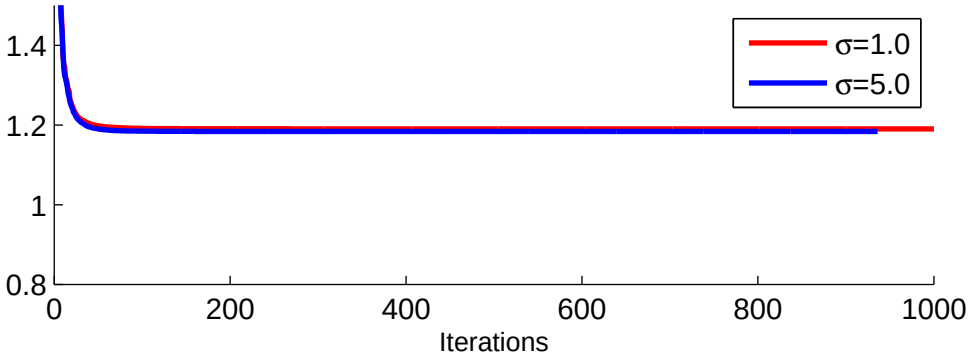


Figure 9.42: Topology: T6, $n = 10000$

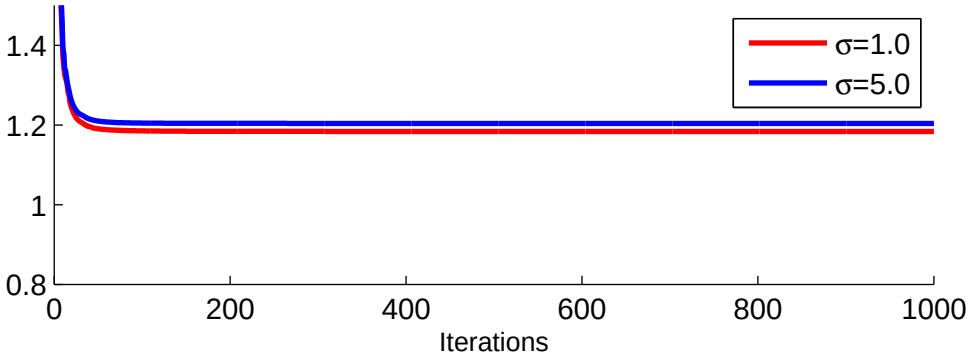


Figure 9.43: Topology: T1, $n = 100000$

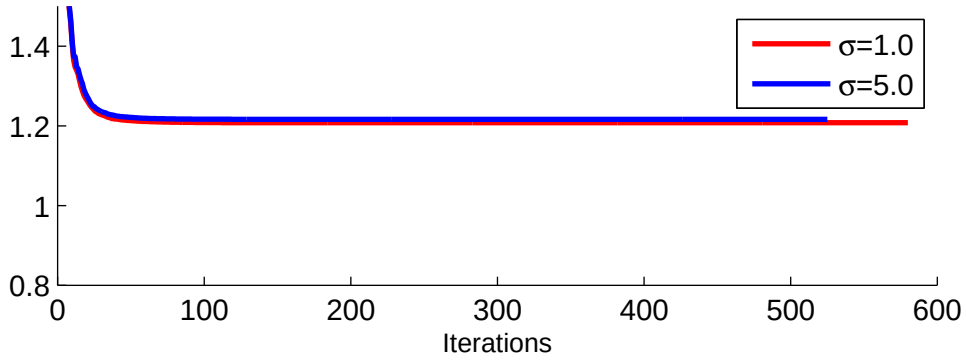


Figure 9.44: Topology: T2, $n = 100000$

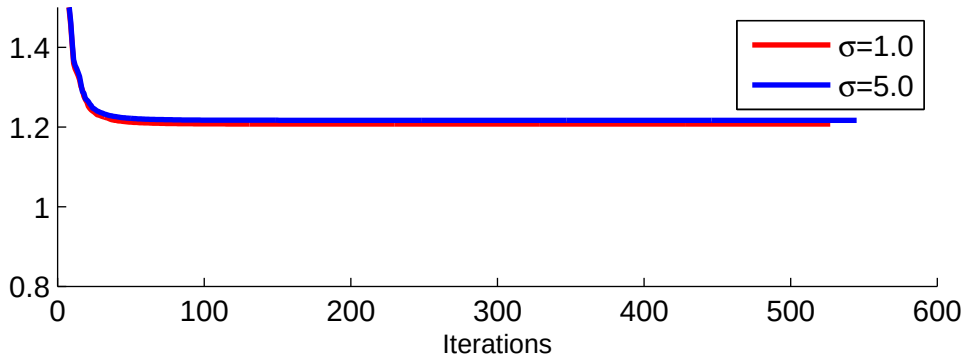


Figure 9.45: Topology: T3, $n = 100000$

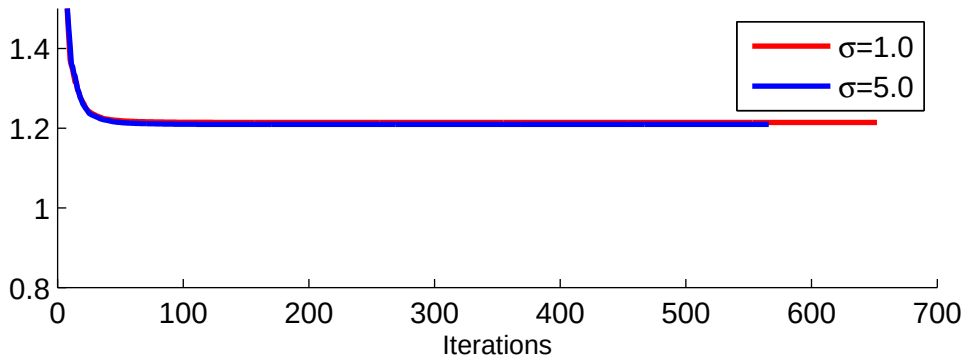


Figure 9.46: Topology: T4, $n = 100000$

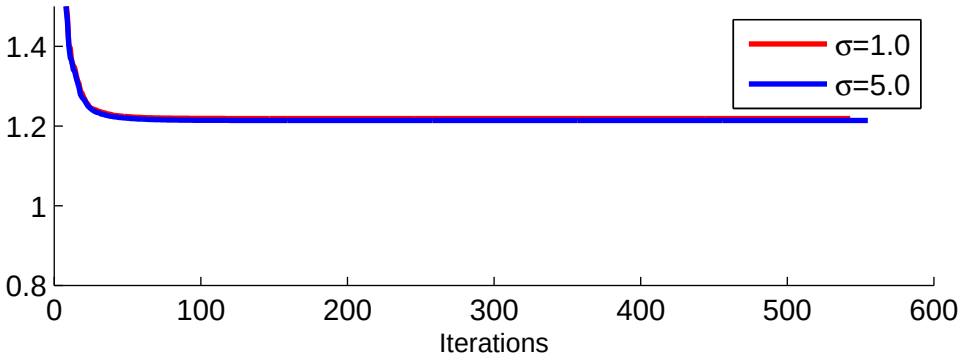


Figure 9.47: Topology: T5, $n = 100000$

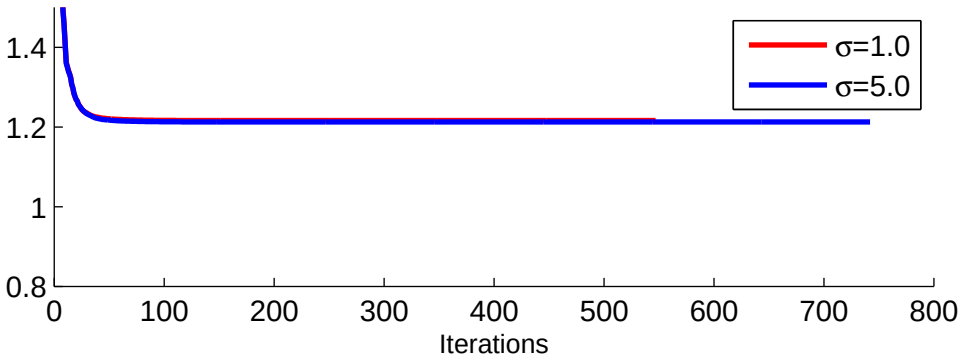
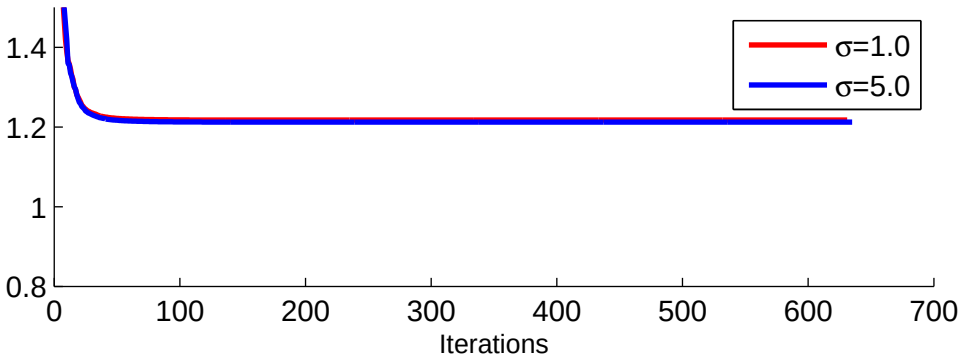


Figure 9.48: Topology: T6, $n = 100000$



Distribution of feature weights (dependent on connectivity distribution T)

Figure 9.49: Graph size $n = 1000$, $\sigma = 1.0$

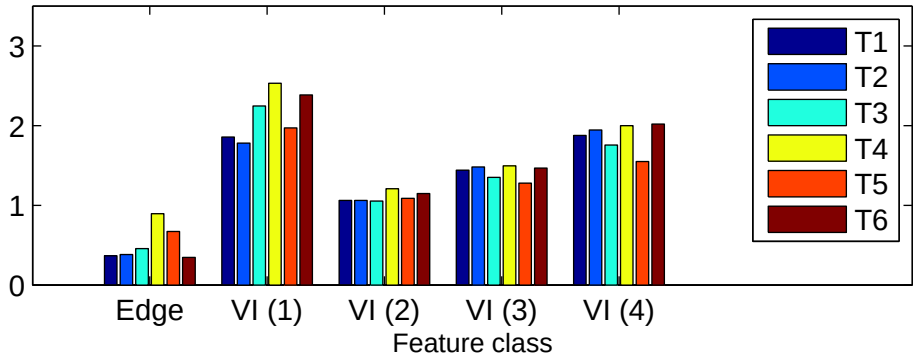


Figure 9.50: Graph size $n = 10000$, $\sigma = 1.0$

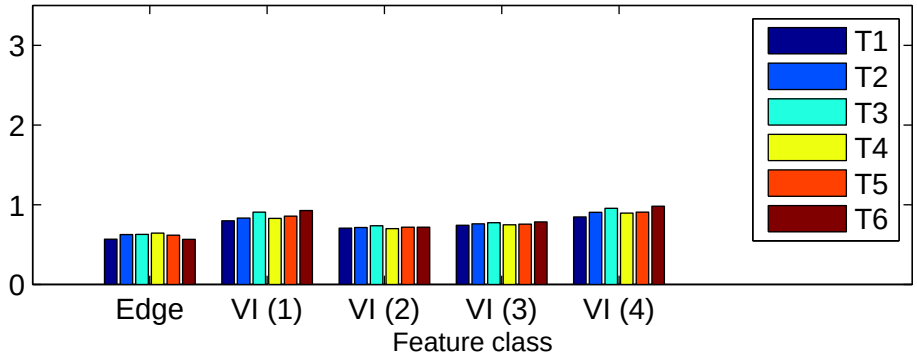
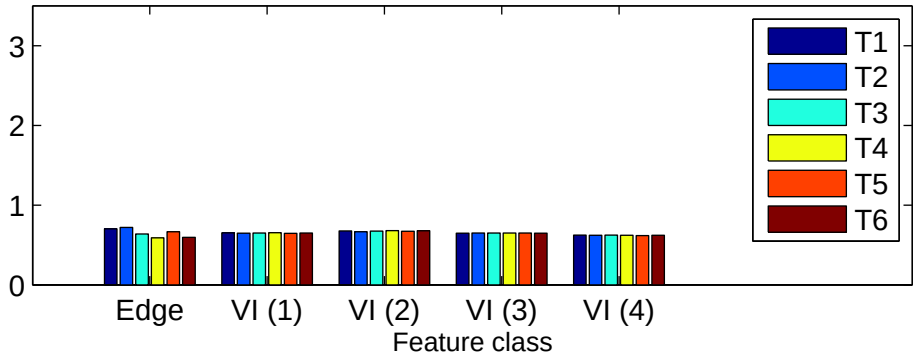


Figure 9.51: Graph size $n = 100000$, $\sigma = 1.0$



Convergence of LBFGS (dependent on connectivity distribution T)

Figure 9.52: Graph size $n = 1000$, $\sigma = 1.0$

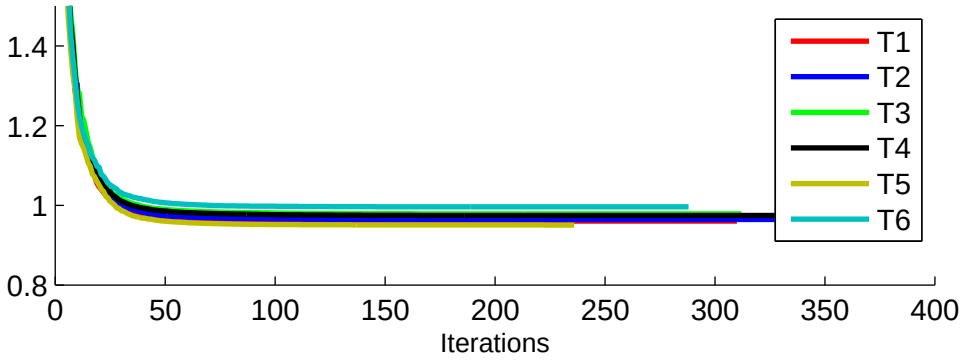


Figure 9.53: Graph size $n = 10000$, $\sigma = 1.0$

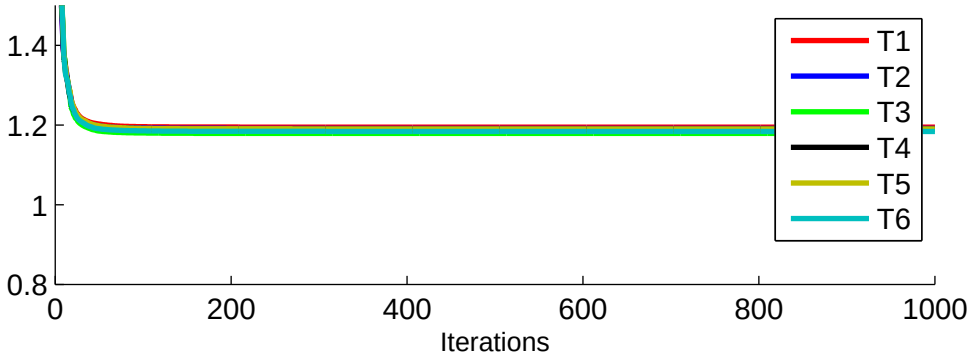
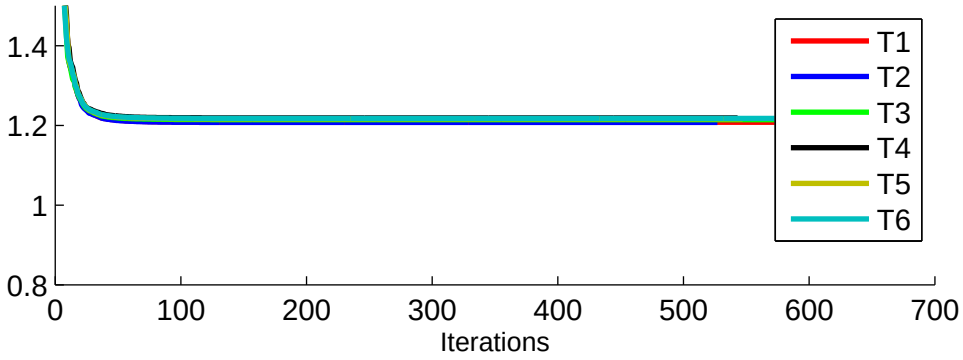


Figure 9.54: Graph size $n = 100000$, $\sigma = 1.0$



9.2 Artificial Weights

Table 9.2: Simulation parameter setup

Simulated graph	
Topology setup	$T1$
Number of vertices	$\{1000, 10000, 100000\}$
Street length distribution (σ)	1.0
Weight distributions	$\{W1, \ldots, W_4\}$
Optimization	
Objective function	$\{O1, O2\}$
Algorithm	L-BFGS
Linesearch	Wolfe-Backtracking, fixed parameters (c1,c2,minStep,...)
Start-values	$N(0, 1)$

The following three graphs show the convergence for the negative log-pseudolikelihood as objective function (O0, as in 6.1.4).

Figure 9.55: Topology: T1, $n = 1000$, $\sigma = 1.0$, variable weights (W1-W4)

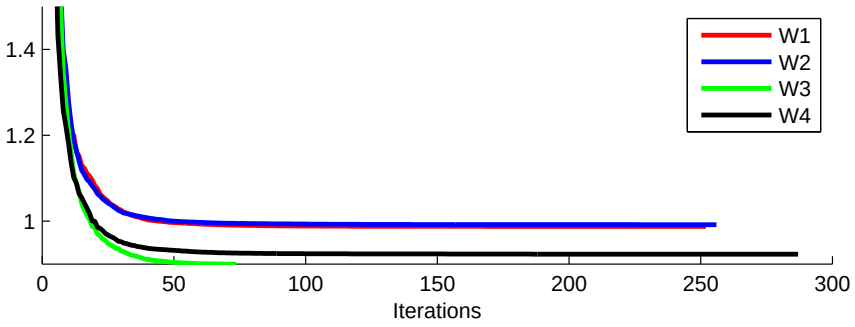


Figure 9.56: Topology: T1, $n = 10000$, $\sigma = 1.0$, variable weights (W1-W4)

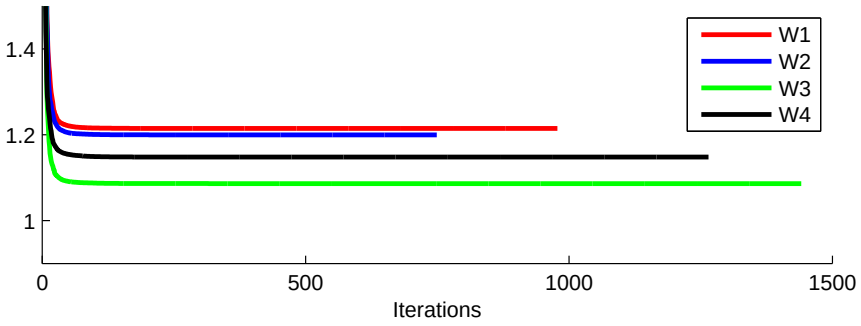
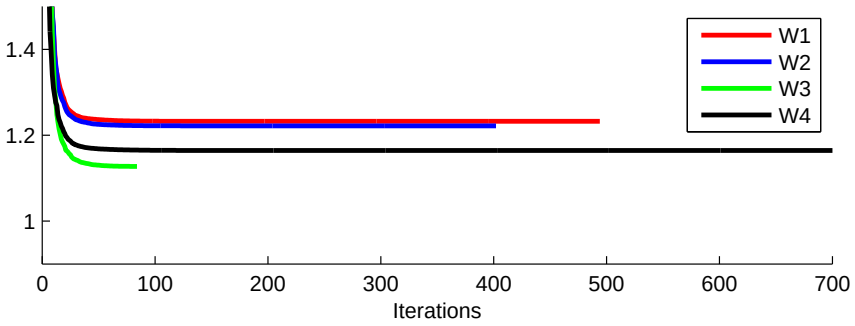


Figure 9.57: Topology: T1, $n = 100000$, $\sigma = 1.0$, variable weights (W1-W4)



The following three graphs show the convergence for the negative log-pseudolikelihood with L1-Penalty as objective function (O2, as in 6.1.4,$\lambda = 2$).

Figure 9.58: Topology: T1, $n = 1000$, $\sigma = 1.0$, variable weights (W1-W4)

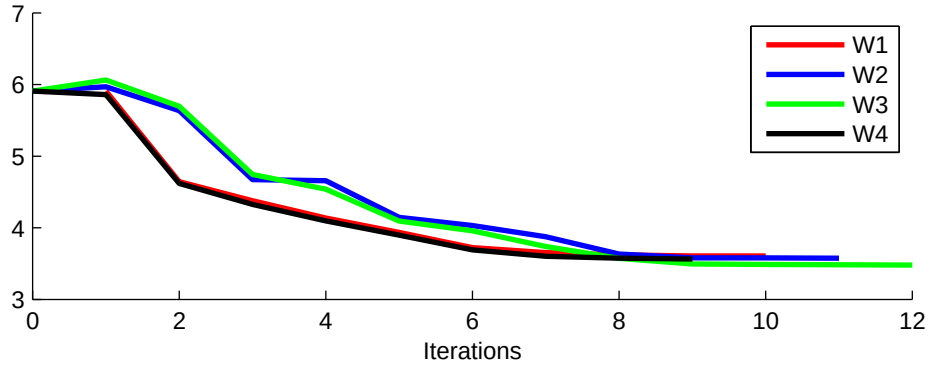


Figure 9.59: Topology: T1, $n = 10000$, $\sigma = 1.0$, variable weights (W1-W4)

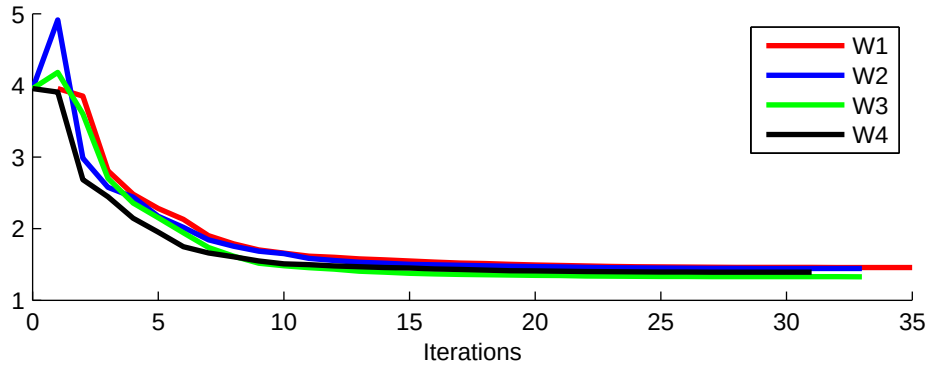
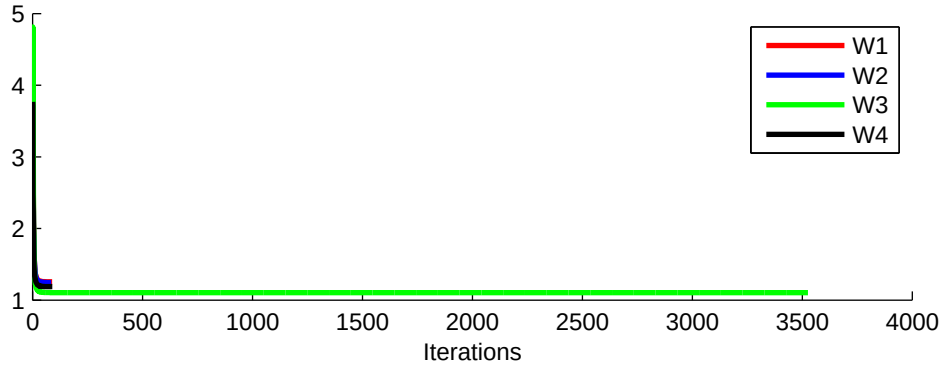


Figure 9.60: Topology: T1, $n = 100000$, $\sigma = 1.0$, variable weights (W1-W4)



The following three graphs show the convergence for the setup: Topology: T1, $n = 10000$, $\sigma = 1.0$, variable Weights (W1-W4) and variable objective function (O1-O5, $\lambda = 1$)

Figure 9.61: Convergence of objective O0, Topology: T1, $n = 10000$, $\sigma = 1.0$, variable Weights (W1-W4)

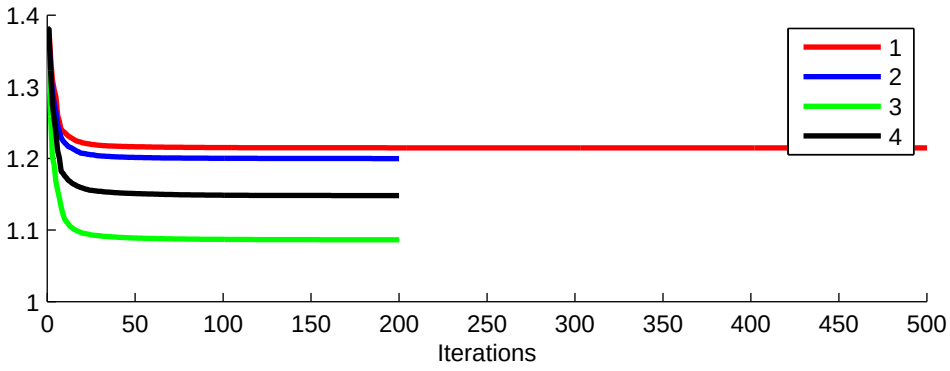


Figure 9.62: Convergence of objective O1,Topology: T1, $n = 10000$, $\sigma = 1.0$, variable Weights (W1-W4)

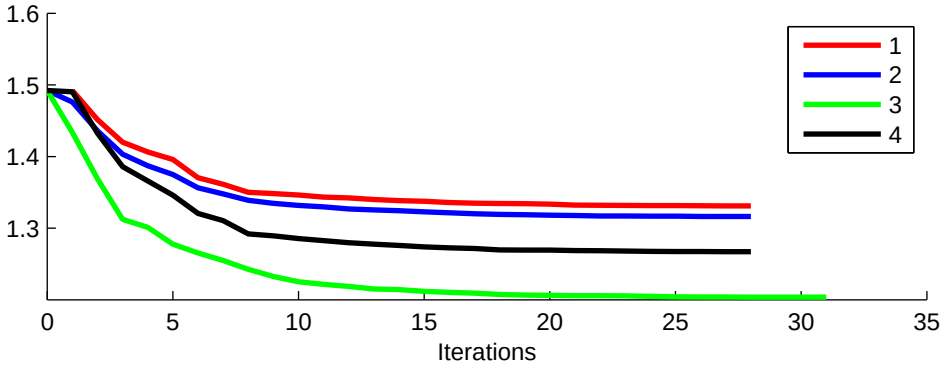


Figure 9.63: Convergence of objective O2,Topology: T1, $n = 10000$, $\sigma = 1.0$, variable Weights (W1-W4)

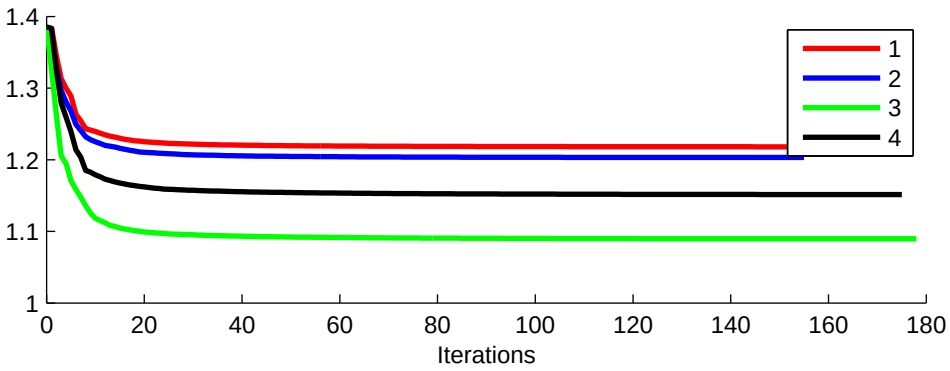


Figure 9.64: Convergence of objective O3,Topology: T1, $n = 10000$, $\sigma = 1.0$, variable Weights (W1-W4)

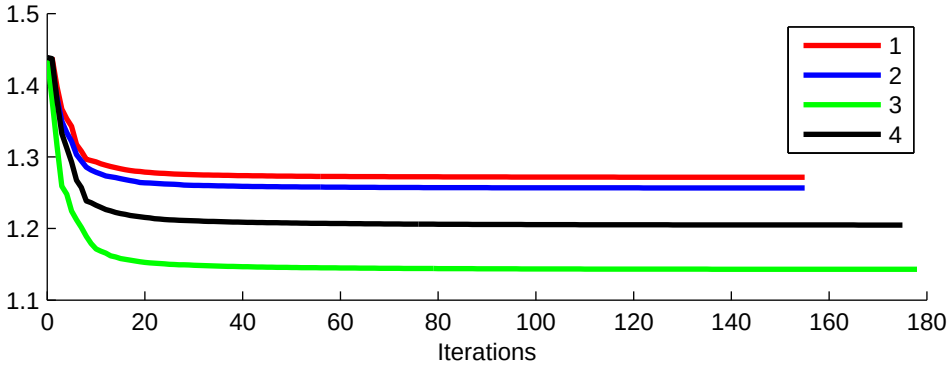


Figure 9.65: Convergence of objective O4,Topology: T1, $n = 10000$, $\sigma = 1.0$, variable Weights (W1-W4)

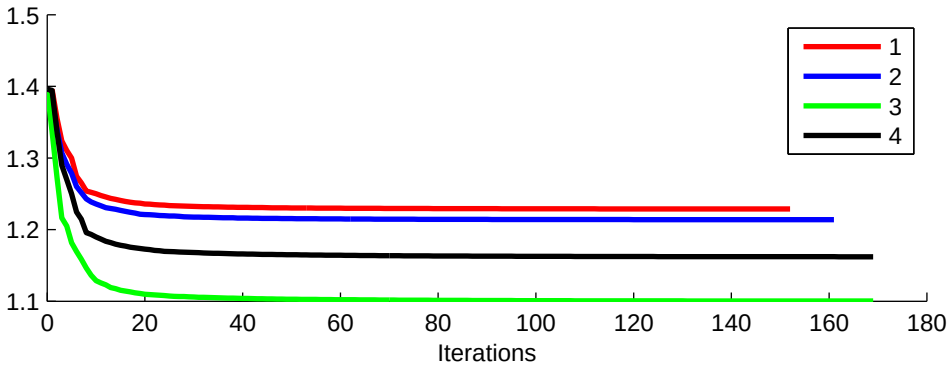
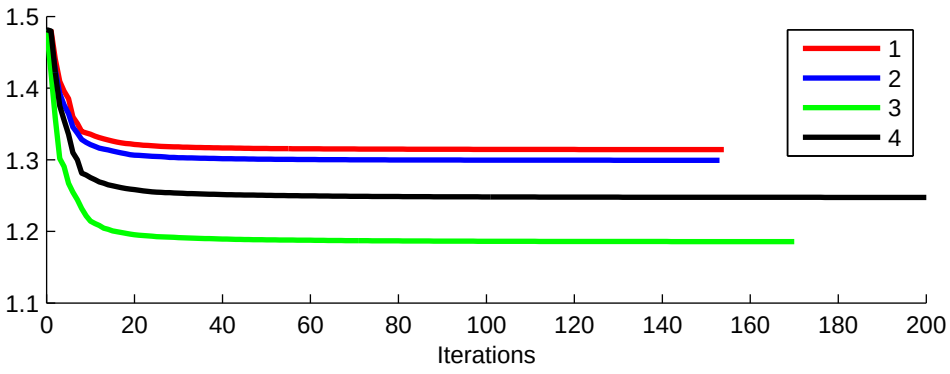


Figure 9.66: Convergence of objective O5,Topology: T1, $n = 10000$, $\sigma = 1.0$, variable Weights (W1-W4)



9.3 Stochastic Gradient Simulations

Table 9.3: Simulation parameter setup

Simulated graph	
Topology setup	$T1$
Number of vertices	1000
Street length distribution (σ)	1.0
Weight distributions	$W1$
Optimization	
Objective function	$O1$
Algorithm	L-BFGS,SGD
Linesearch	L-BFGS with Wolfe-Backtr.,
	fixed params.(c1,c2,minStep,...)
	SGD Robbins-Monro, decay param. $\alpha \in \{0.5, 0.75, 1.0\}$
	SGD with fixed steps $\delta \in \{0.001, 0.01, 0.1\}$
Batchsizes	{5,10,50,100}
Start-values	$N(0, 1)$

Robbins-Monro-Decay

The following graphs show the results for the negative log-pseudolikelihood using the SGD with different RM-decays and batchsizes

Figure 9.67: Decay param: $\alpha = 0.5$, different batch sizes

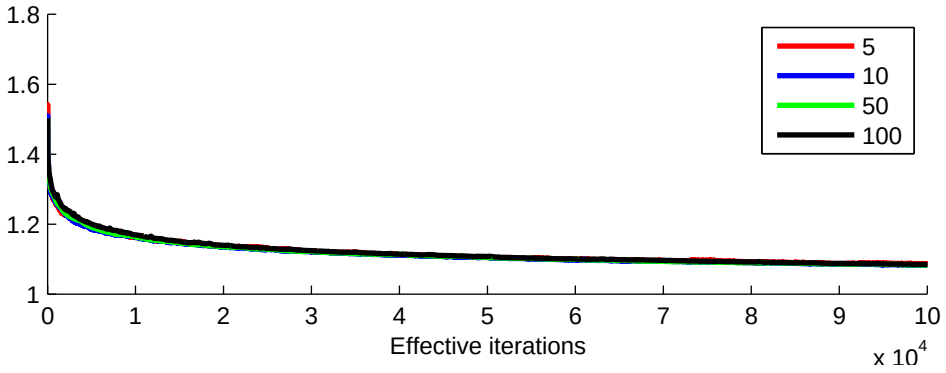


Figure 9.68: Decay param: $\alpha = 0.75$, different batch sizes

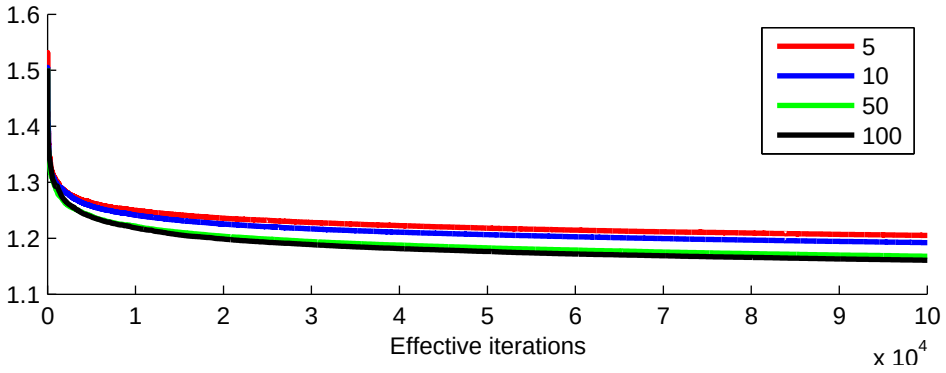
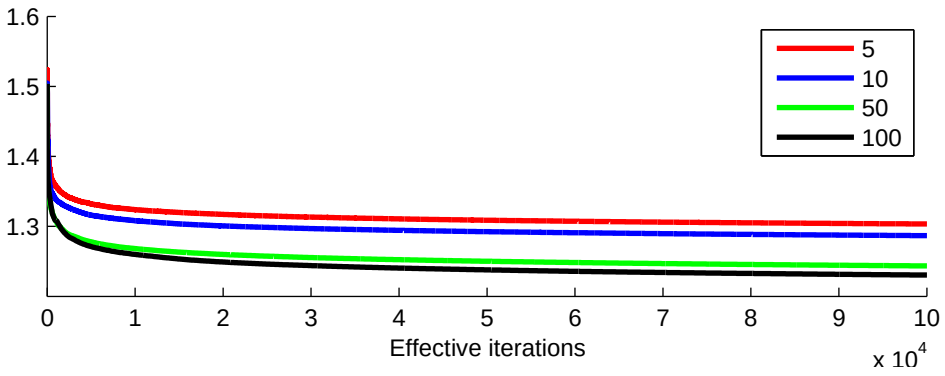


Figure 9.69: Decay param: $\alpha = 1$, different batch sizes



Fixed step sizes

The following graphs show the results for the negative log-pseudolikelihood using the SGD with different fixed stepsizes and batchsizes

Figure 9.70: Fixed step size: $\delta = 0.001$, different batch sizes

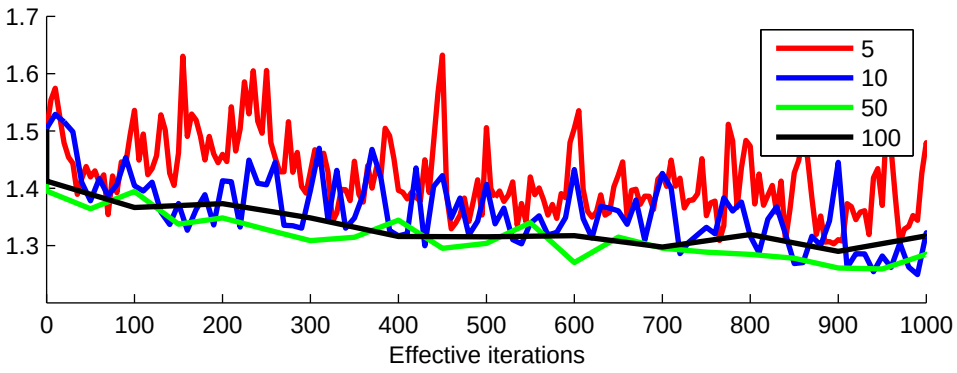


Figure 9.71: Fixed step size: $\delta = 0.01$, different batch sizes

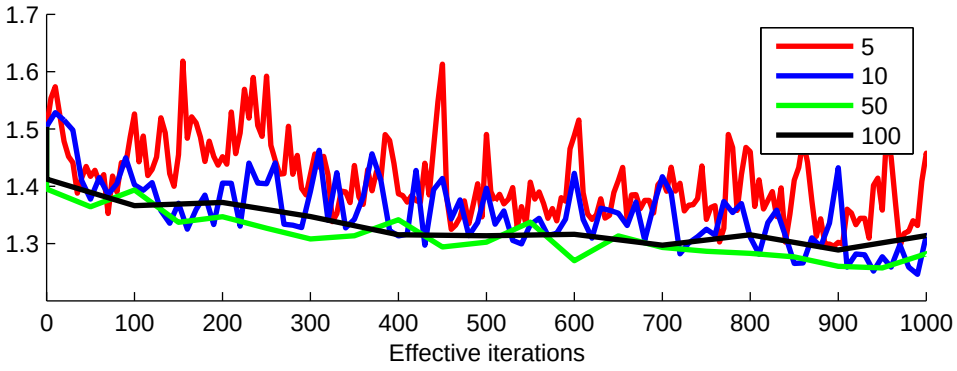
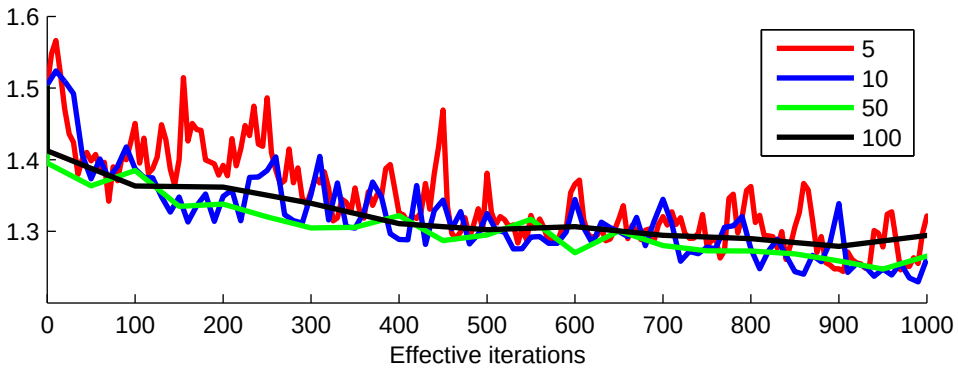


Figure 9.72: Fixed step size: $\delta = 0.1$, different batch sizes



9.4 Contrastive Divergence

Table 9.4: Simulation parameter setup

Simulated graph	
Topology setup	$T1$
Number of vertices	10000
Street length distribution (σ)	1.0
Weight distributions	W1
Optimization	
Algorithm	Contrastive Divergence {naive and enhanced} with {1,10} MCMC runs
Batchsize	{10,50,100}
Stepsize	{0.001,0.03125,0.1,0.5}
Start-values	$N(0,1)$

Small stepsizes

Figure 9.73: Stepsize: 0.001 Batchsize: 10

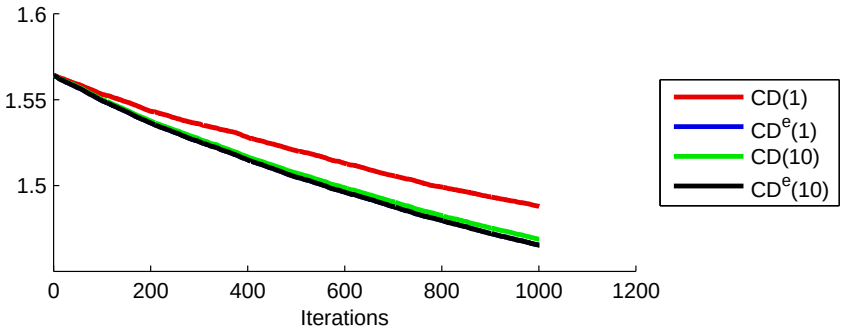


Figure 9.74: Stepsize: 0.001 Batchsize: 50

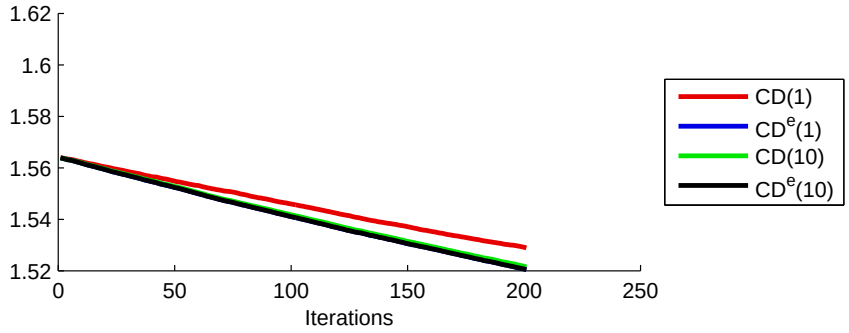
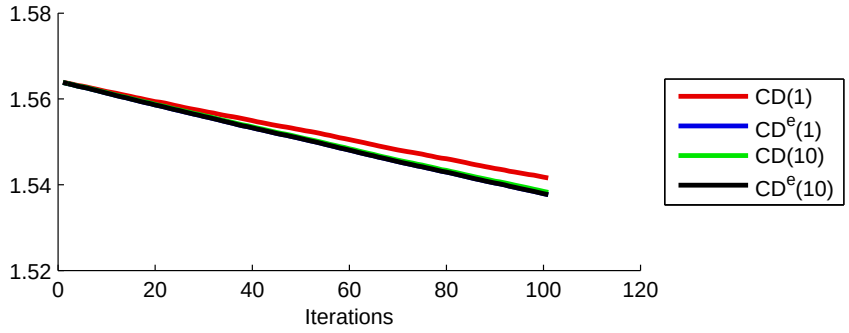


Figure 9.75: Stepsize: 0.001 Batchsize: 100



Medium stepsizes

Figure 9.76: Stepsize: 0.03125 Batchsize: 10

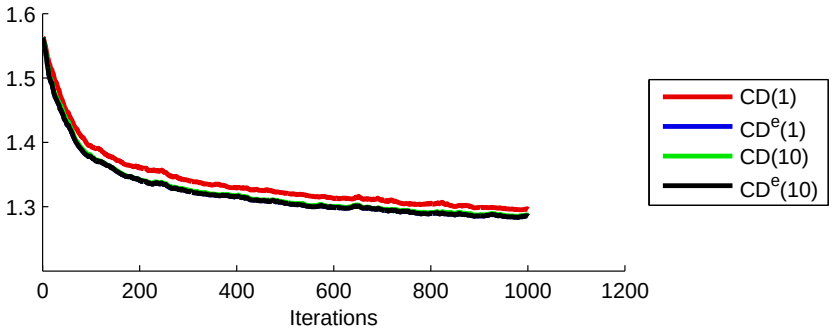


Figure 9.77: Stepsize: 0.03125 Batchsize: 50

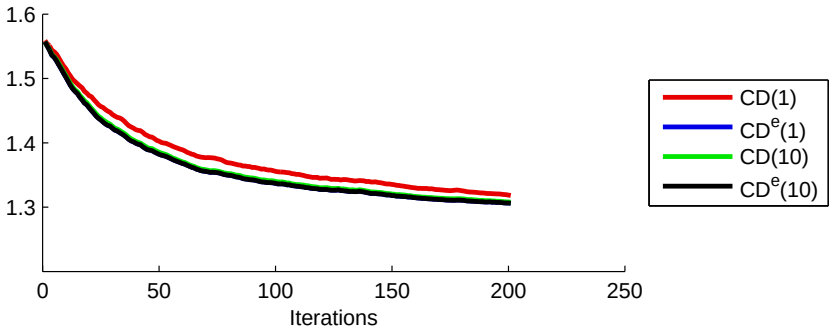


Figure 9.78: Stepsize: 0.03125 Batchsize: 100

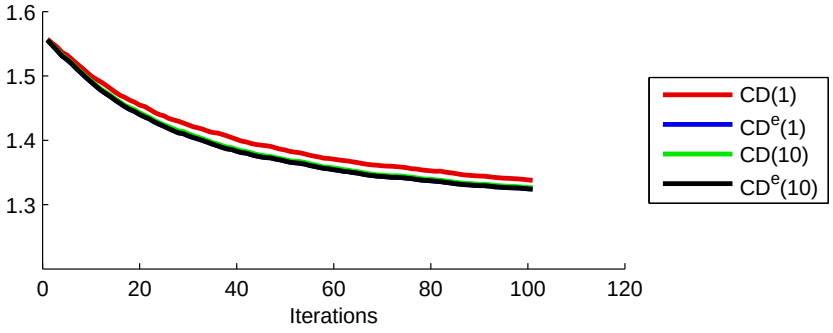


Figure 9.79: Stepsize: 0.1 Batchsize: 10

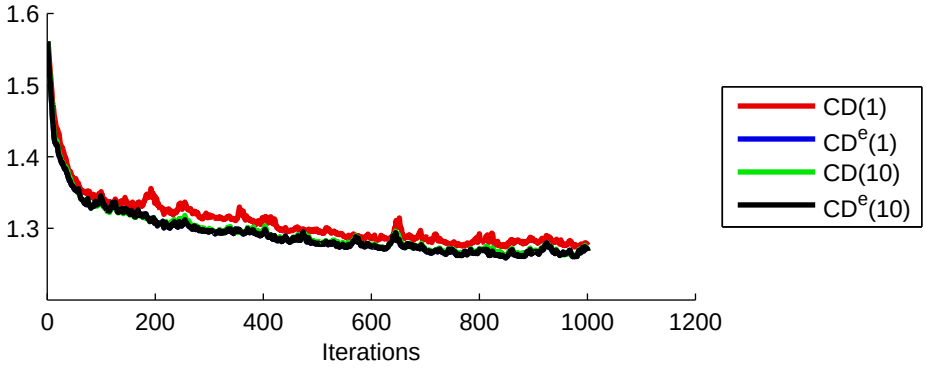


Figure 9.80: Stepsize: 0.1 Batchsize: 50

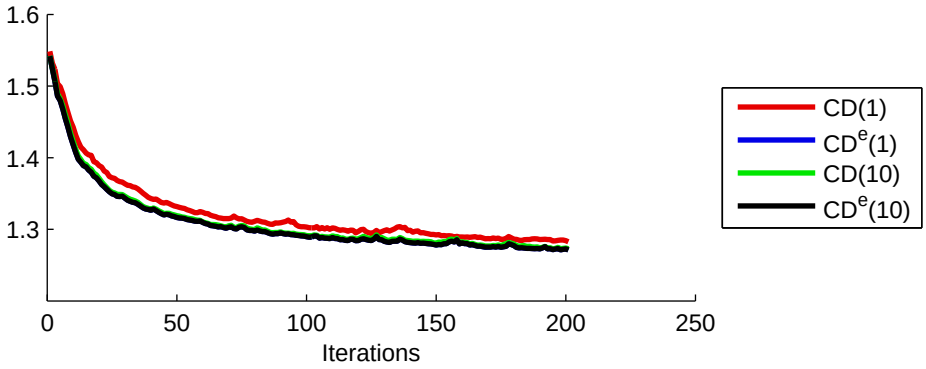
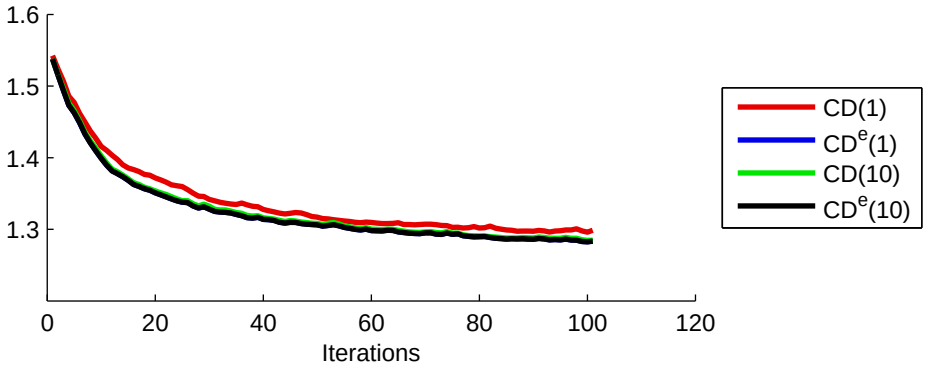


Figure 9.81: Stepsize: 0.1 Batchsize: 100



Big stepsizes

Figure 9.82: Stepsize: 0.5 Batchsize: 10

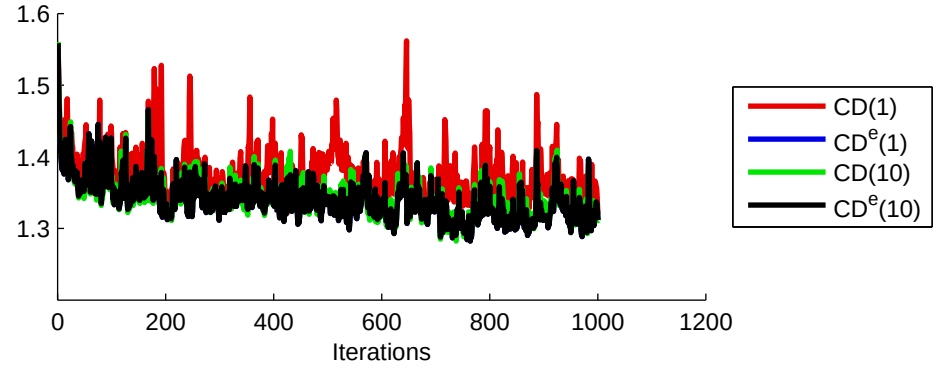


Figure 9.83: Stepsize: 0.5 Batchsize: 50

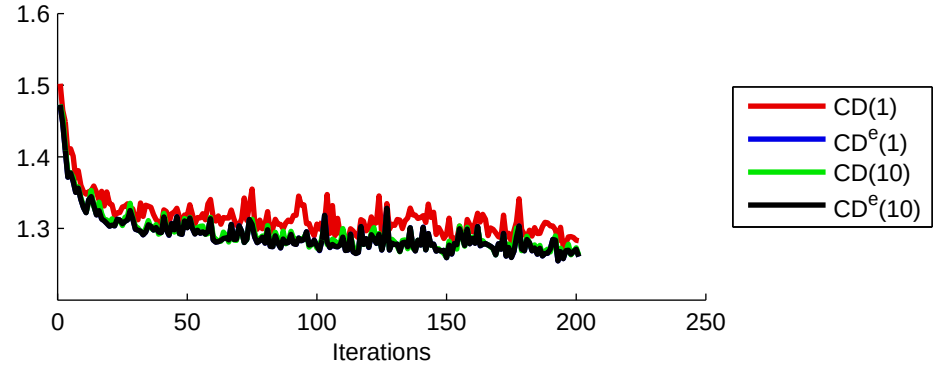
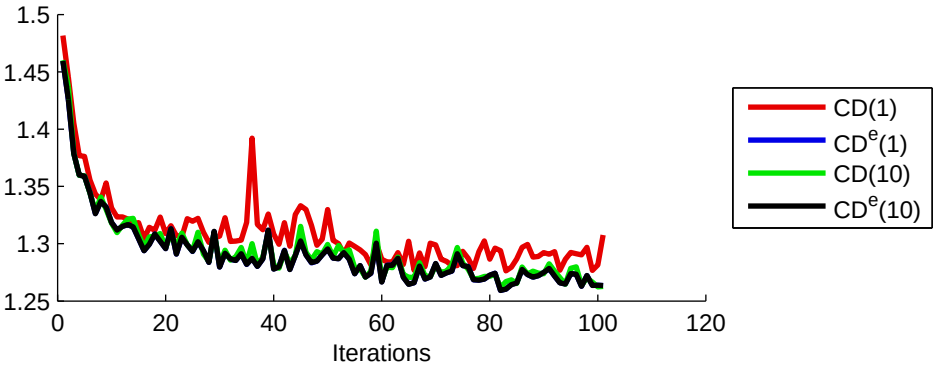


Figure 9.84: Stepsize: 0.5 Batchsize: 100



Stochastic Average Gradient with Ringbuffer

Table 9.5: Simulation parameter setup

Simulated graph	
Topology setup	$T1$
Number of vertices	10000
Street length distribution (σ)	1.0
Weight distributions	$W1$
Optimization	
Objective function	$O1$
Algorithm	{SAG, SGD}
Batch Size	{5,10,50,100}
Linesearch	Fixed steps $\delta \in \{0.001, 0.01, 0.1\}$
Start-values	$N(0, 1)$

Batch Size 5

Figure 9.85: Step size: $\delta = 0.001$

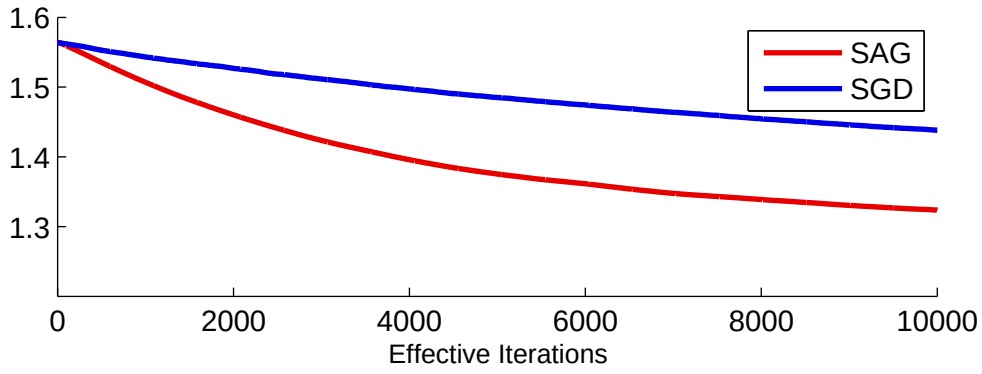


Figure 9.86: Step size: $\delta = 0.01$

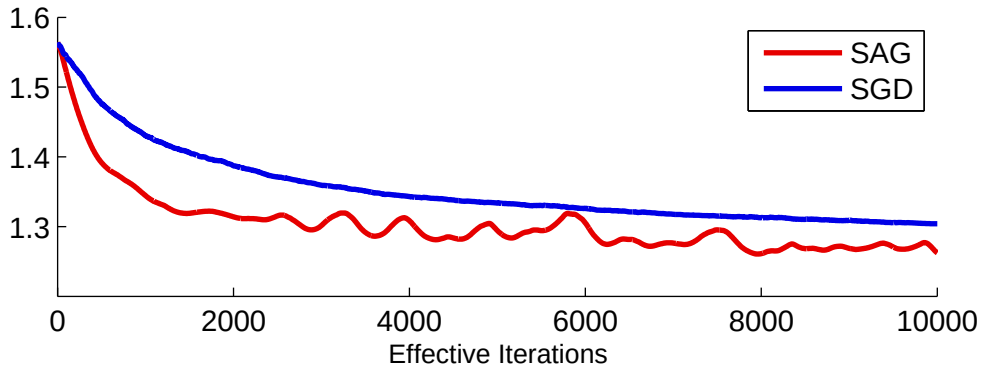
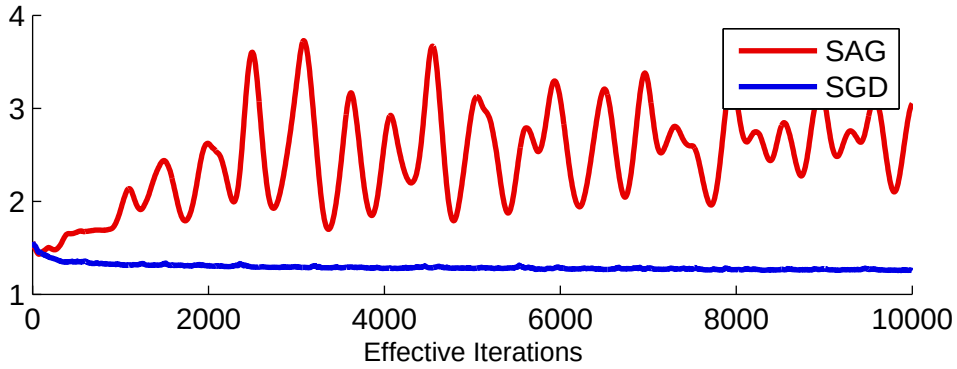


Figure 9.87: Step size: $\delta = 0.1$



Batch Size 10

Figure 9.88: Step size: $\delta = 0.001$

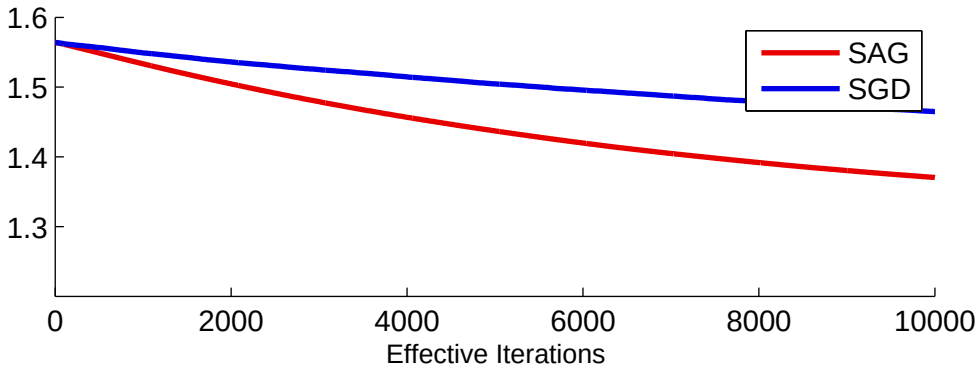


Figure 9.89: Step size: $\delta = 0.01$

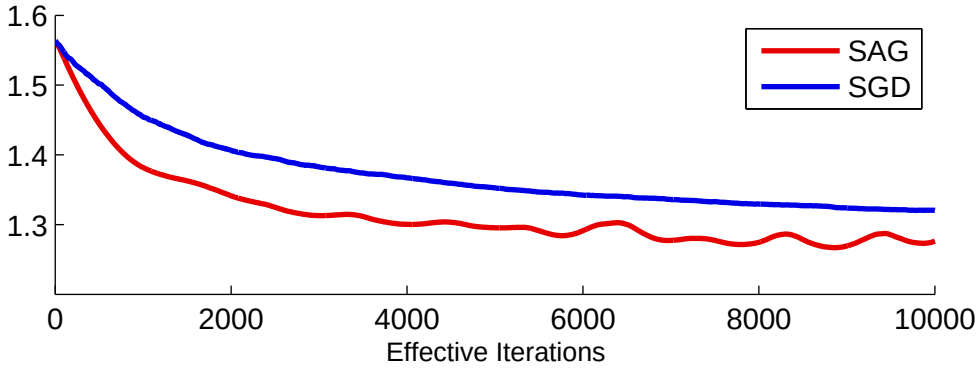
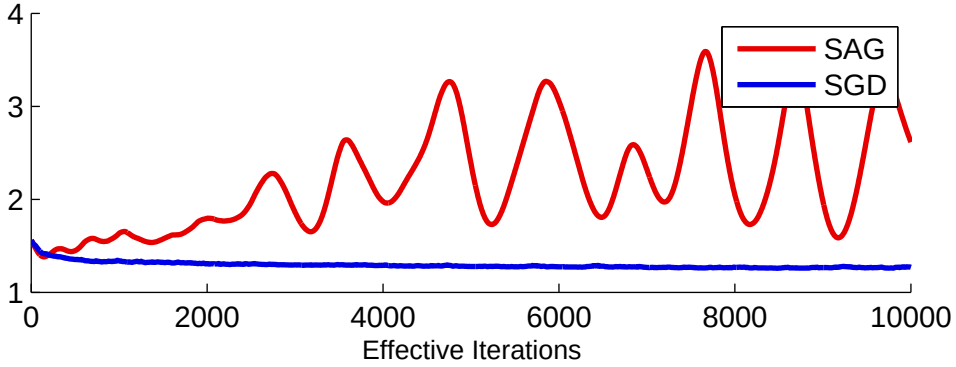


Figure 9.90: Step size: $\delta = 0.1$



Batch Size 50

Figure 9.91: Step size: $\delta = 0.001$

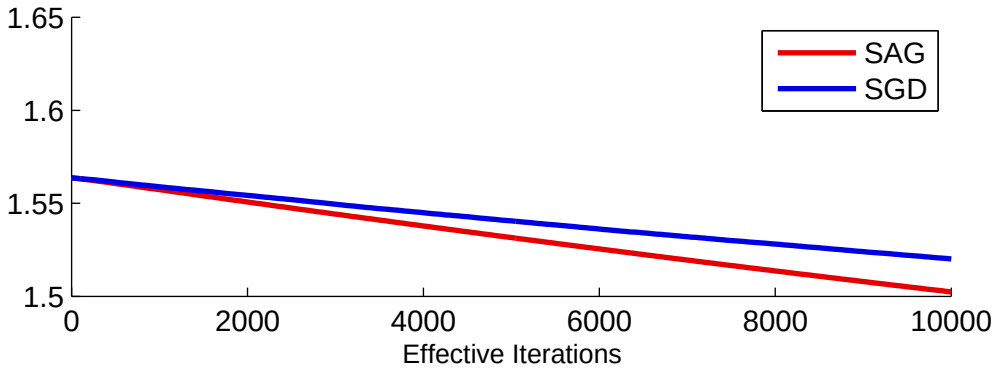


Figure 9.92: Step size: $\delta = 0.01$

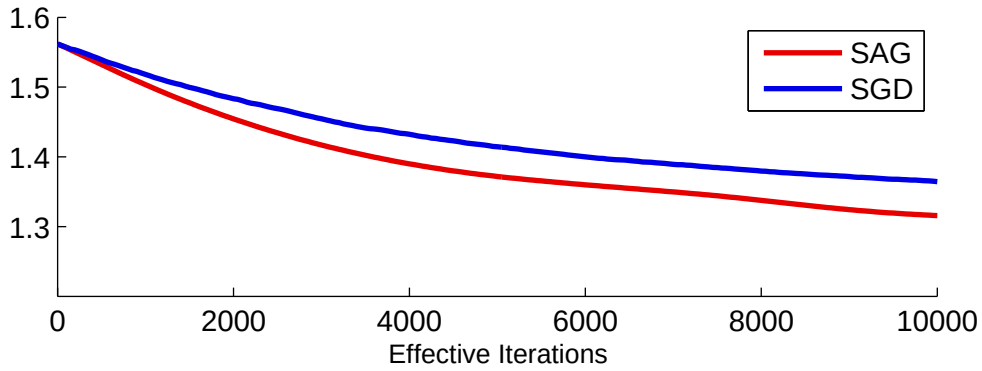
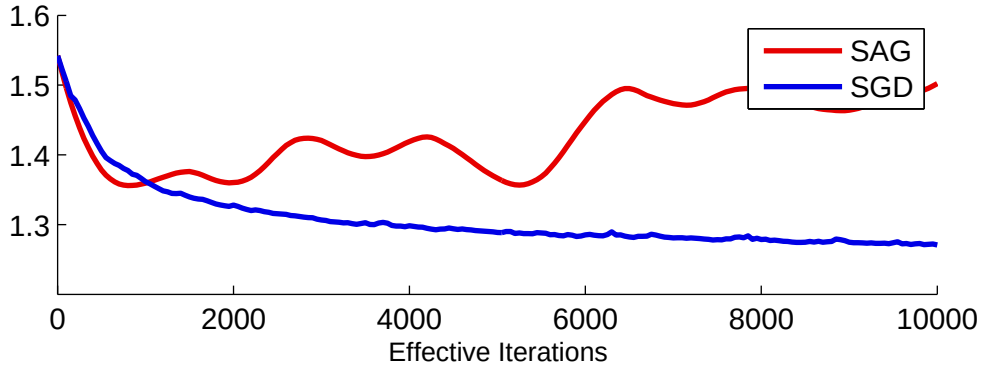


Figure 9.93: Step size: $\delta = 0.1$



Batch Size 100

Figure 9.94: Step size: $\delta = 0.001$

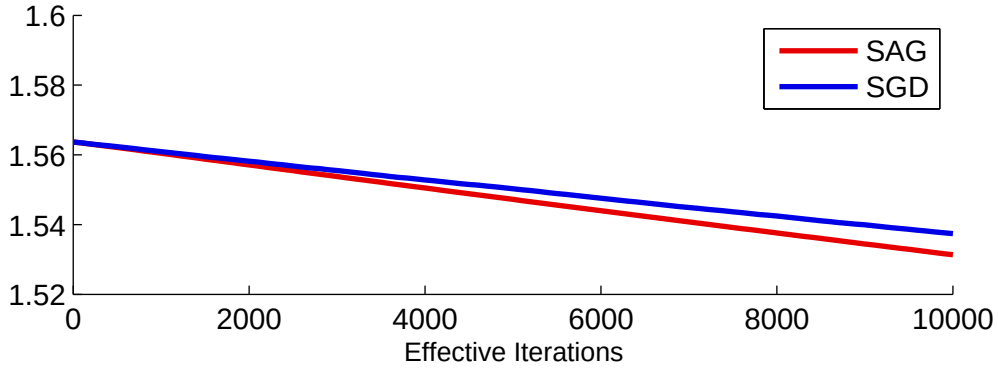


Figure 9.95: Step size: $\delta = 0.01$

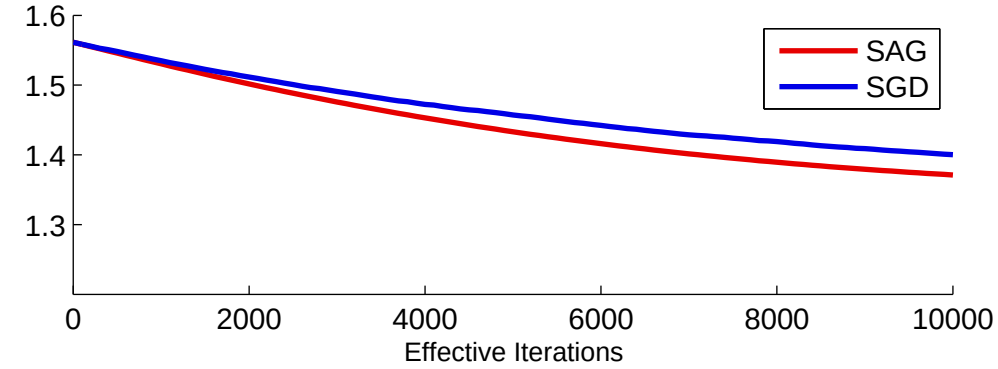
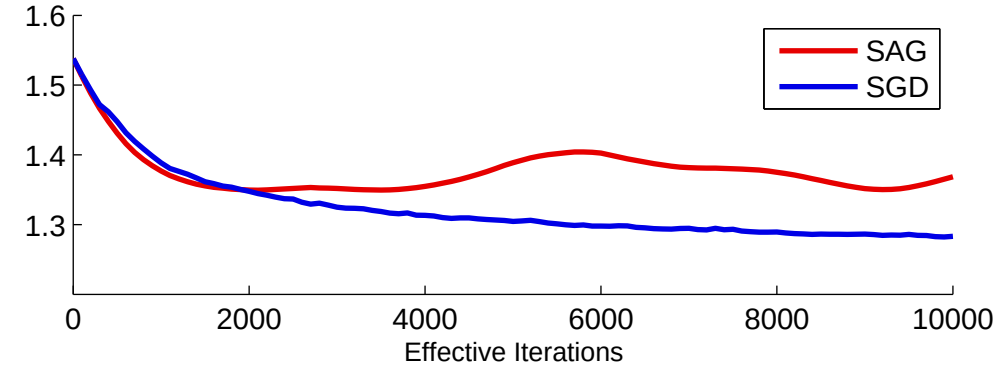


Figure 9.96: Step size: $\delta = 0.1$



9.5 SAG Variants

Table 9.6: Simulation parameter setup

Simulated graph	
Topology setup	$T1$
Number of vertices	{10000}
Street length distribution (σ)	1.0
Weight distributions	$W1$
Optimization	
Objective function	$O1$
Algorithm	SAG with Ringbuffer (SAG)
	SAG with weighting: (SAG-W1)
	SAG with weighting:(SAG-W2)
	SAG with local weighting (5.4.4) (SAG-LW)
	SAG with robust weighting (5.4.5) (SAG-RW)
	SAG with robust local weighting (5.4.6)(SAG-RLW)
Linesearch	Constant Step Sizes: {0.01,0.1,1}
Batchsize	{10,50,100}
Start-values	$N(0, 1)$

9.5.1 Stepsize 0.001

Figure 9.97: Stepsize 0.001 BatchSize 10

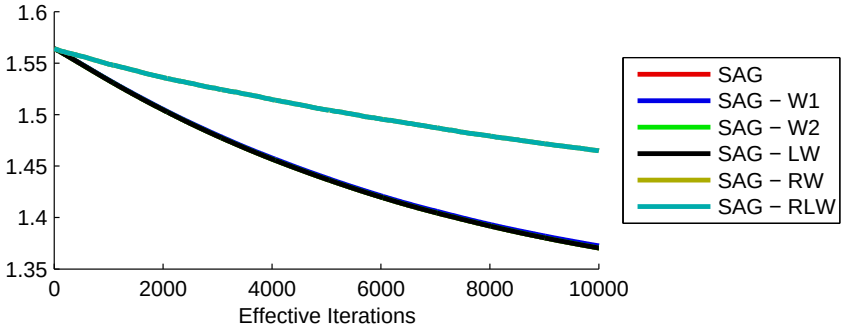


Figure 9.98: Stepsize 0.001 BatchSize 50

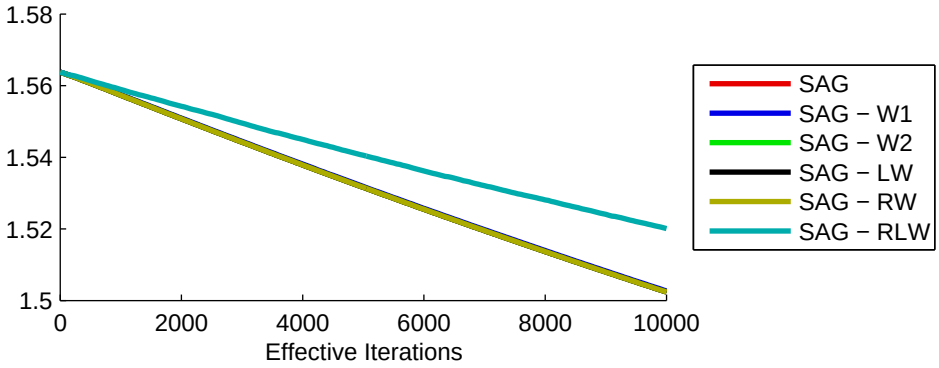
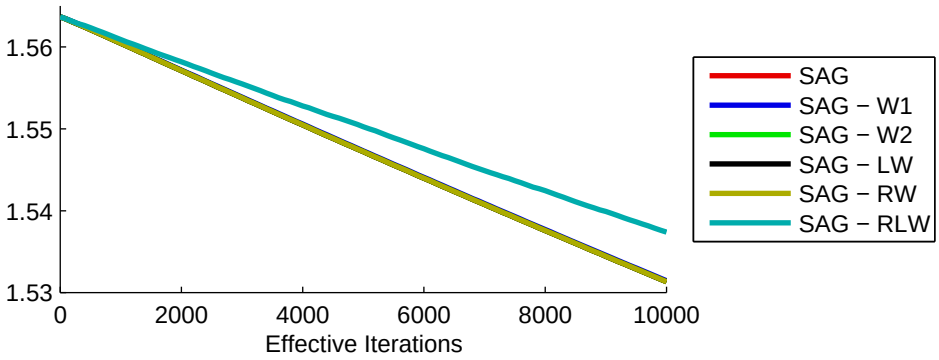


Figure 9.99: Stepsize 0.001 BatchSize 100



9.5.2 Stepsize 0.01

Figure 9.100: Stepsize 0.01 BatchSize 10

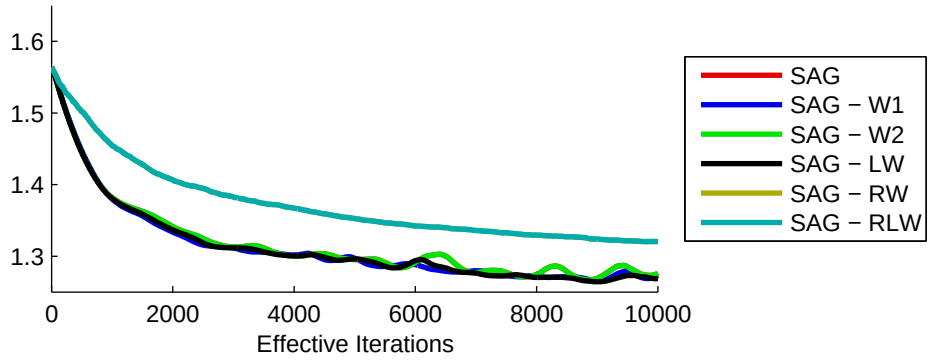


Figure 9.101: Stepsize 0.01 BatchSize 50

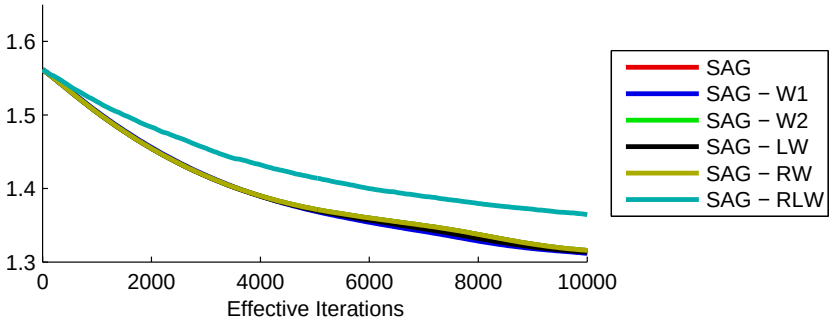
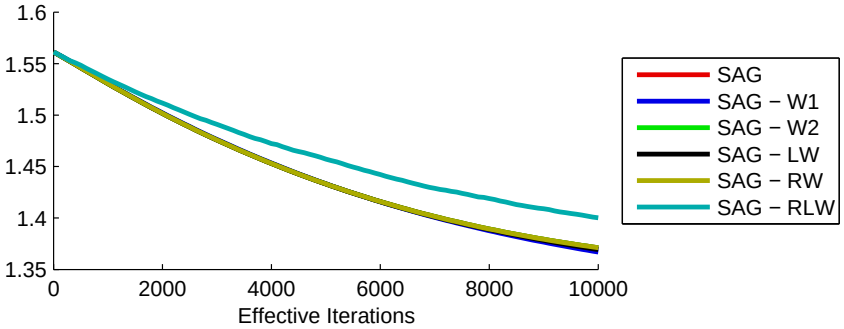


Figure 9.102: Stepsize 0.01 BatchSize 100



9.5.3 Stepsize 0.03125

Figure 9.103: Stepsize 0.03125 BatchSize 10

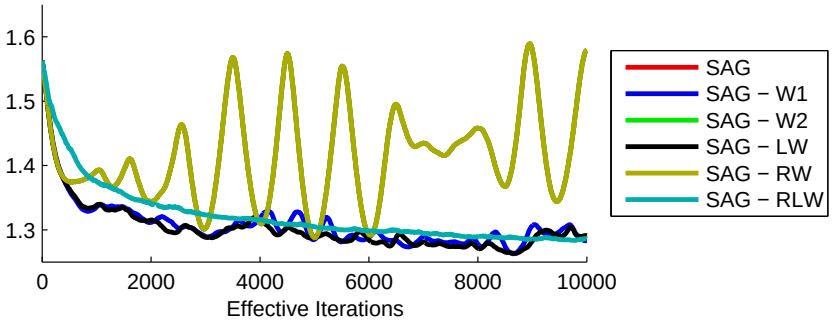


Figure 9.104: Stepsize 0.03125 BatchSize 50

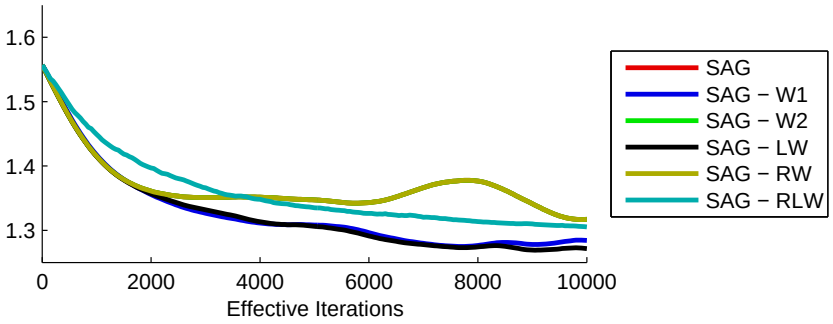
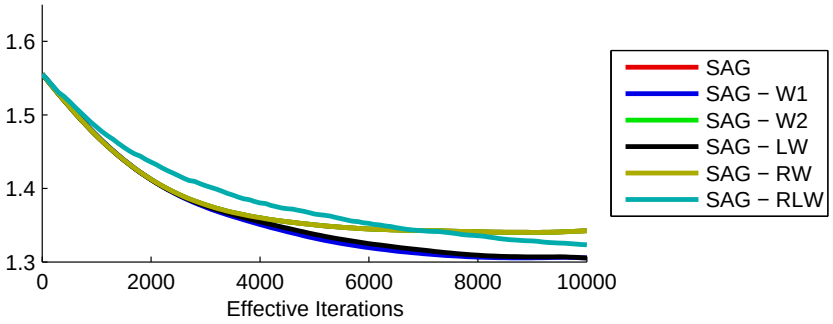


Figure 9.105: Stepsize 0.03125 BatchSize 100



9.5.4 Stepsize 0.1

Figure 9.106: Stepsize 0.1 BatchSize 10

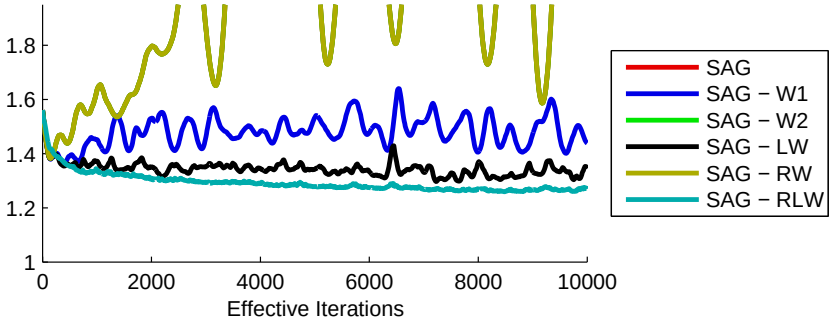


Figure 9.107: Stepsize 0.1 BatchSize 50

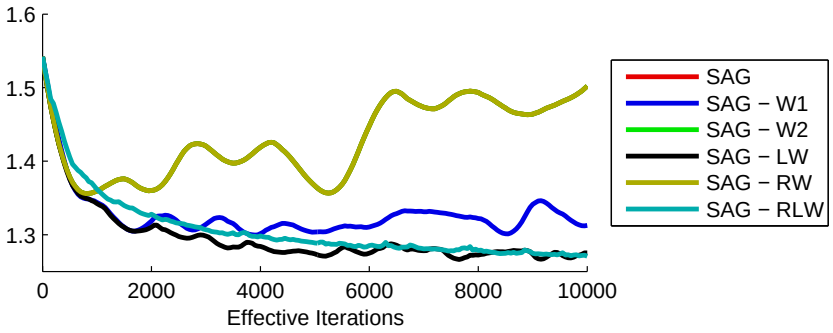
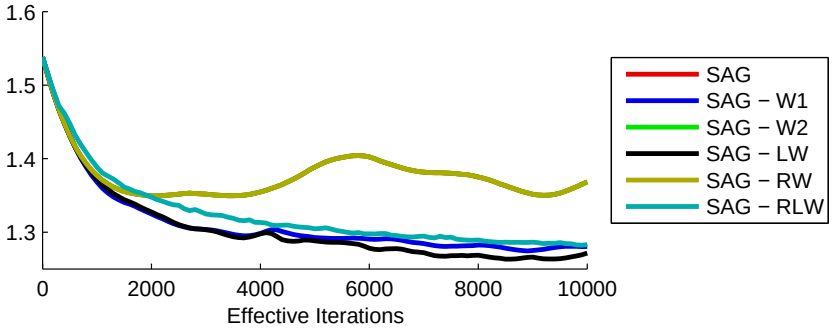


Figure 9.108: Stepsize 0.1 BatchSize 100



9.5.5 Stepsize 0.5

Figure 9.109: Stepsize 0.5 BatchSize 10

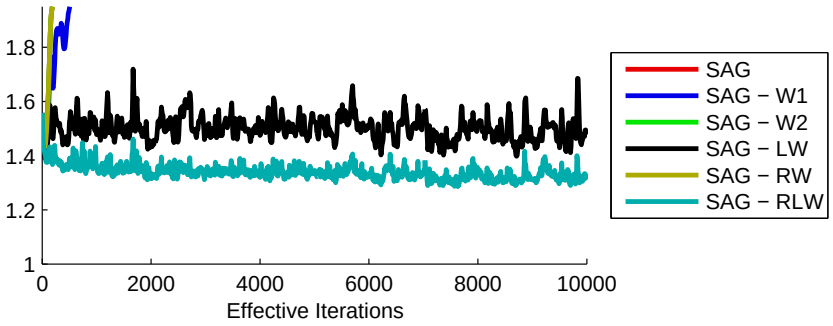


Figure 9.110: Stepsize 0.5 BatchSize 50

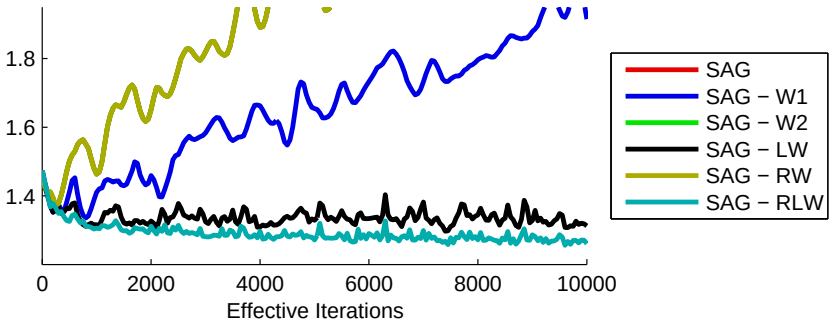
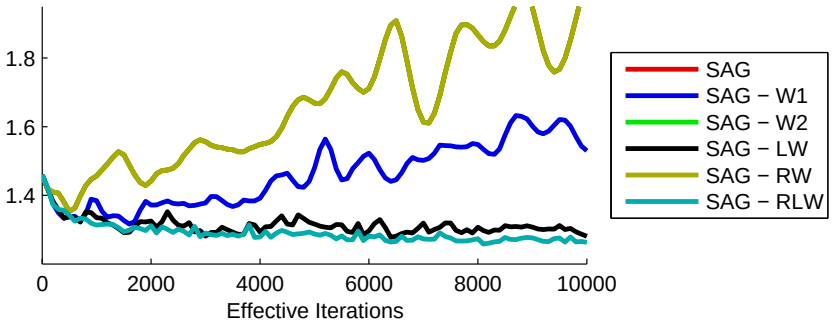


Figure 9.111: Stepsize 0.5 BatchSize 100



9.6 Locally Robust Weighted Stochastic Average Gradient

Table 9.7: Simulation parameter setup

Simulated graph	
Topology setup	$T1$
Number of vertices	$\{1000, 10000, 100000\}$
Street length distribution (σ)	1.0
Weight distributions	$W1$
Optimization	
Objective function	$O1$
Algorithm	SAG with robust local weighting (5.4.6)(SAG-RLW)
Linesearch	Fixed step sizes $\{0.1, 0.2, 0.5, 1, 2\}$
Batchsize	{10,50,100}
Start-values	$N(0, 1)$

Small Graphs

Figure 9.112: $n = 1000$, batch size 10, different step sizes

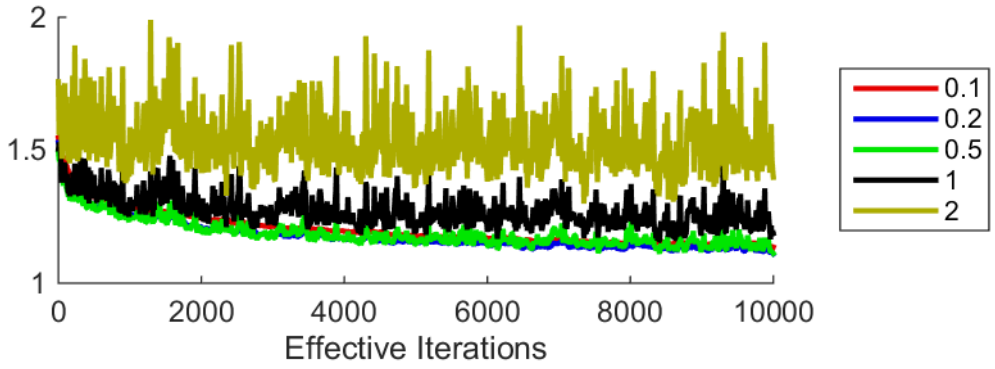


Figure 9.113: $n = 1000$, batch size 50, different step sizes

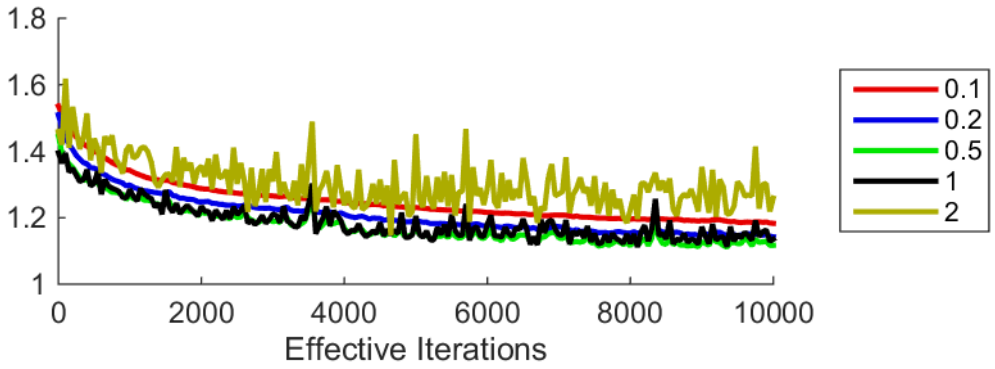
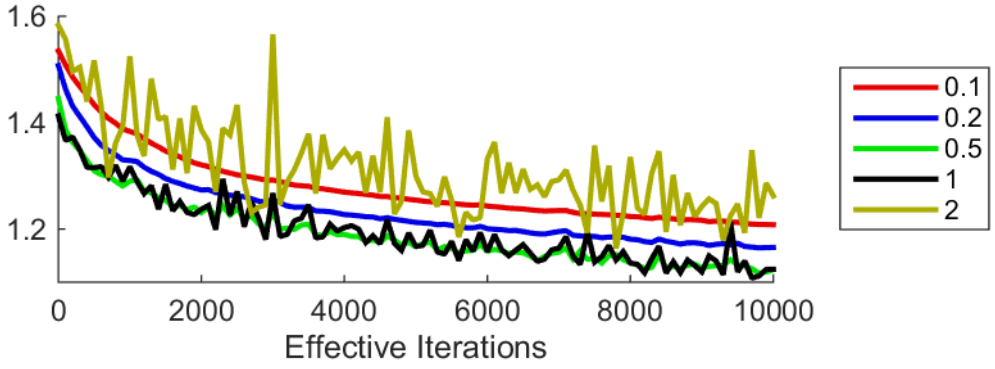


Figure 9.114: $n = 1000$, batch size 100, different step sizes



Medium Graphs

Figure 9.115: $n = 10000$, batch size 10, different step sizes

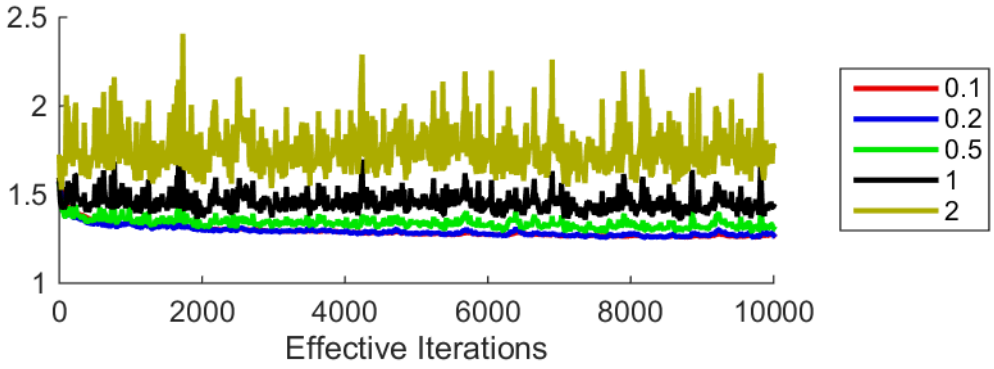


Figure 9.116: $n = 10000$, batch size 50, different step sizes

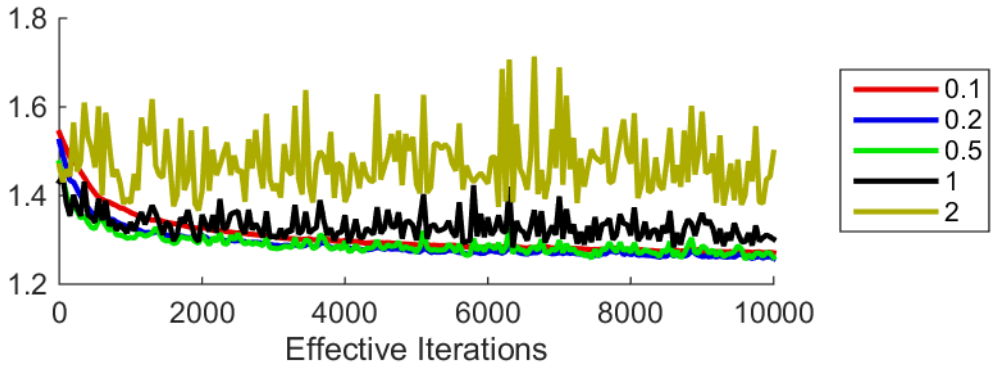
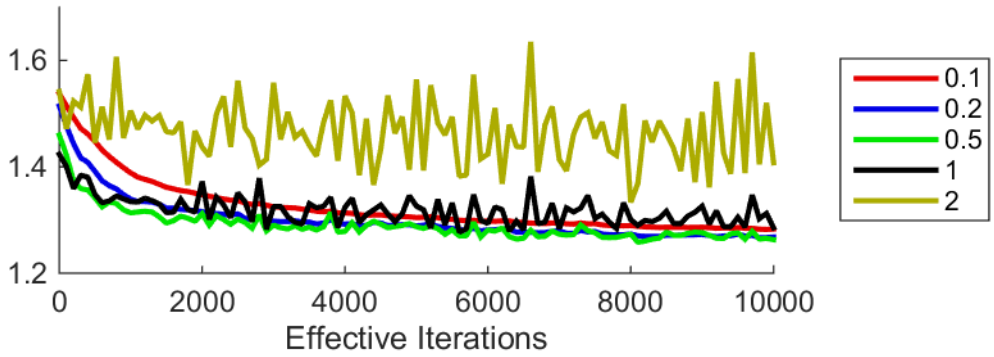


Figure 9.117: $n = 10000$, batch size 100, different step sizes



Large Graphs

Figure 9.118: $n = 100000$, batch size 10, different step sizes

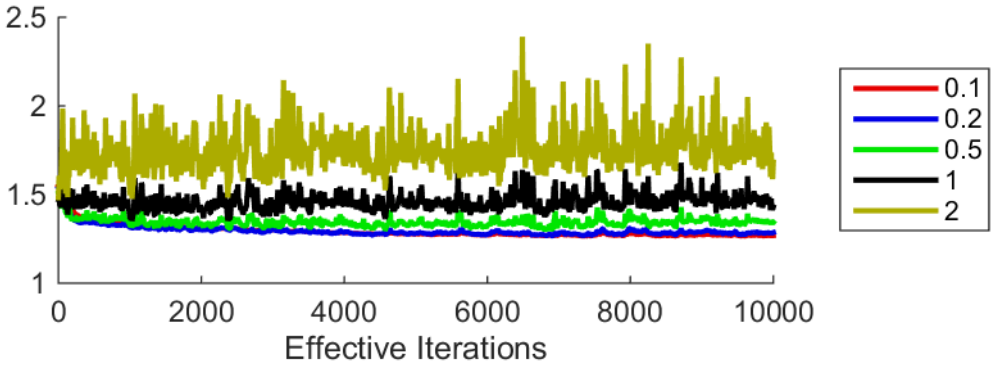


Figure 9.119: $n = 100000$, batch size 50, different step sizes

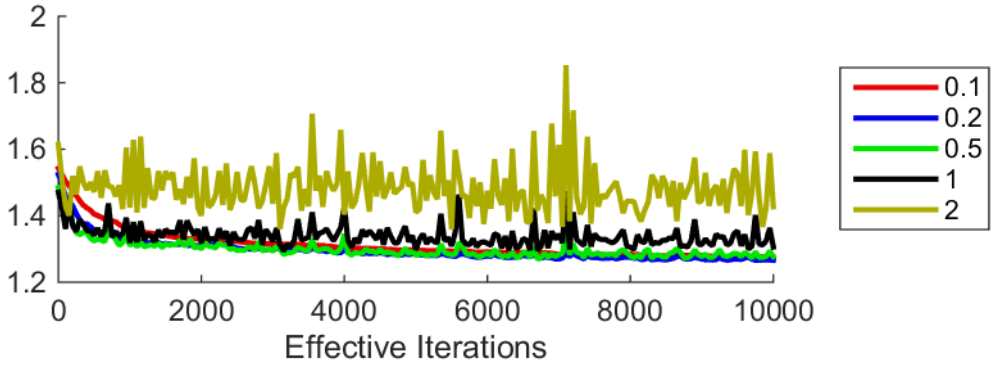
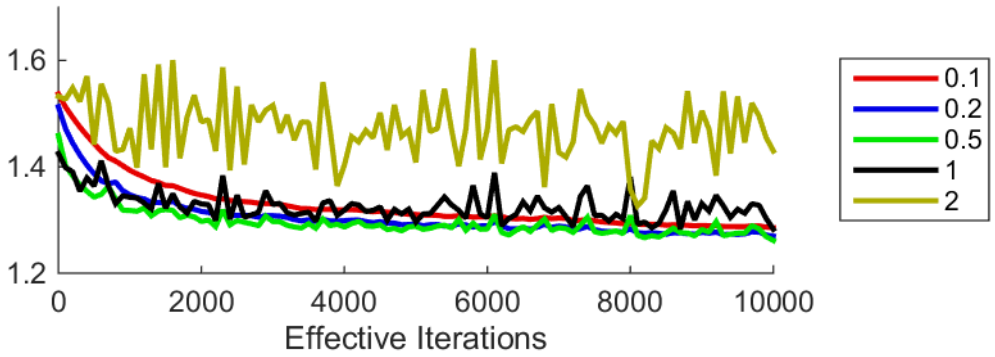


Figure 9.120: $n = 100000$, batch size 100, different step size



Bibliography

[AG11] Asmussen, Soeren ; Glynn, Peter W.: A new proof of convergence of {MCMC} via the ergodic theorem. In: *Statistics & Probability Letters* 81 (2011), Nr. 10, 1482 - 1485. http://dx.doi.org/http://dx.doi.org/10.1016/j.spl.2011.05.004. – DOI http://dx.doi.org/10.1016/j.spl.2011.05.004. – ISSN 0167–7152

[Bes75] Besag, Julian: Statistical Analysis of Non-Lattice Data. In: *Journal of the Royal Statistical Society. Series D (The Statistician)* 24 (1975), September, Nr. 3, 179–195. http://dx.doi.org/10.2307/2987782. – DOI 10.2307/2987782. – ISSN 0039–0526. – ArticleType: research-article / Full publication date: Sep., 1975 / Copyright © 1975 Royal Statistical Society

[BL05] Bottou, Léon ; LeCun, Yann: On-line Learning for Very Large Datasets. In: *Applied Stochastic Models in Business and Industry* 21 (2005), Nr. 2, 137-151. http://leon.bottou.org/papers/bottou-lecun-2004a

[Blu54] Blum, Julius R.: Multidimensional Stochastic Approximation Methods. In: *The Annals of Mathematical Statistics* 25 (1954), Dezember, Nr. 4, 737–744. http://projecteuclid.org/euclid.aoms/1177728659. – ISSN 0003–4851, 2168–8990. – Mathematical Reviews number (MathSciNet) MR65092, Zentralblatt MATH identifier0056.38305

[Bro70] Broyden, C. G.: The Convergence of a Class of Double-rank Minimization Algorithms 1. General Considerations. In: *IMA Journal of Applied Mathematics* 6 (1970), Januar, Nr. 1, 76–90. http://dx.doi.org/10.1093/imamat/6.1.76. – DOI 10.1093/imamat/6.1.76. – ISSN 0272–4960, 1464–3634

[DAB04] Dietterich, Thomas G. ; Ashenfelter, Adam ; Bulatov, Yaroslav: Training Conditional Random Fields via Gradient Tree Boosting. In: *Proceedings of the Twenty-first International Conference on Machine Learning*. New York, NY, USA : ACM, 2004 (ICML '04). – ISBN 1–58113–838–5, 28–

[EHJT04] Efron, Bradley ; Hastie, Trevor ; Johnstone, Iain ; Tibshirani, Robert: Least angle regression. In: *The Annals of Statistics* 32 (2004), Nr. 2, 407–499. http://dx.doi.org/10.1214/009053604000000067. – DOI 10.1214/009053604000000067. – ISSN 0090–5364, 2168–8966

[GG84] Geman, Stuart ; Geman, D.: Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images. In: *IEEE Transactions on*

Pattern Analysis and Machine Intelligence PAMI-6 (1984), Nr. 6, S. 721–741. http://dx.doi.org/10.1109/TPAMI.1984.4767596. – DOI 10.1109/TPAMI.1984.4767596. – ISSN 0162–8828

[Gol70] Goldfarb, Donald: A Family of Variable-Metric Methods Derived by Variational Means. In: *Mathematics of Computation* 24 (1970), Januar, Nr. 109, 23–26. http://dx.doi.org/10.2307/2004873. – DOI 10.2307/2004873. – ISSN 0025–5718

[GR92] Gelman, Andrew ; Rubin, Donald B.: Inference from Iterative Simulation Using Multiple Sequences. In: *Statistical Science* 7 (1992), November, Nr. 4, 457–472. http://dx.doi.org/10.2307/2246093. – DOI 10.2307/2246093. – ISSN 0883–4237. – ArticleType: research-article / Full publication date: Nov., 1992 / Copyright © 1992 Institute of Mathematical Statistics

[GRS95] Gilks, W. R. ; Richardson, S. ; Spiegelhalter, David: *Markov Chain Monte Carlo in Practice*. CRC Press, 1995. – ISBN 9780412055515

[GS03] Goodman, Joshua ; Soon, Msr-tr-coming: Exponential Priors for Maximum Entropy Models. In: *In Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2003, S. 305–312

[Has70] Hastings, W. K.: Monte Carlo sampling methods using Markov chains and their applications. In: *Biometrika* 57 (1970), Januar, Nr. 1, 97–109. http://dx.doi.org/10.1093/biomet/57.1.97. – DOI 10.1093/biomet/57.1.97. – ISSN 0006–3444, 1464–3510

[HC71] Hammersley, JM ; Clifford, P: *Markov fields on finite graphs and lattices, unpublished*. 1971

[HF00] Haan, Laurens d. ; Ferreira, Ana: *Extreme Value Theory: An Introduction*. S.l. : Springer Verlag Gmbh, 2000. – ISBN 9780817643645

[Hin00] Hinton, Geoffrey: Training Products of Experts by Minimizing Contrastive Divergence. In: *Neural Computation* 14 (2000), S. 2002

[HRF06] Hamze, "Firas ; Rivasseau, Jean-Noel ; Freitas", Nando de: "Information Theory Tools to Rank MCMC Algorithms on Probabilistic Graphical Models". In: *"Information Theory and Applications Workshop (ITA)"*, "2006"

[JPR14] Johannesson, Pär ; Podgórski, Krzysztof ; Rychlik, Igor: Modelling roughness of road profiles on parallel tracks using roughness indicators. In: *Chalmers Publication Library (CPL)* (2014)

[JR13] Johannesson, Pär ; Rychlik, Igor: Laplace Processes for Describing Road Profiles. In: *Procedia Engineering* 66 (2013), 464–473. http://dx.doi.org/10.1016/j.proeng.2013.12.099. – DOI 10.1016/j.proeng.2013.12.099. – ISSN 1877–7058

[JS+14] Johannesson, Paer ; Speckert, Michael u. a.: Guide to Load Analysis for Durability in Vehicle Engeneering, 2014

[KF09] Koller, Daphne ; Friedman, Nir: *Probabilistic Graphical Models: Principles and Techniques - Adaptive Computation and Machine Learning*. The MIT Press, 2009. – ISBN 0262013193, 9780262013192

[KL51] Kullback, S. ; Leibler, R. A.: On Information and Sufficiency. In: *The Annals of Mathematical Statistics* 22 (1951), März, Nr. 1, 79–86. http://dx.doi.org/10.1214/aoms/1177729694. – DOI 10.1214/aoms/1177729694. – ISSN 0003–4851

[Laf01] Lafferty, John: Conditional random fields: Probabilistic models for segmenting and labeling sequence data, Morgan Kaufmann, 2001, S. 282–289

[LCFK07] Liao, Lin ; Choudhury, Tanzeem ; Fox, Dieter ; Kautz, Henry: Training Conditional Random Fields Using Virtual Evidence Boosting. In: *Proceedings of the 20th International Joint Conference on Artifical Intelligence*. San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 2007 (IJCAI'07), 2530–2535

[McC03] McCallum, Andrew: Efficiently Inducing Features of Conditional Random Fields. In: *Proceedings of the Nineteenth Conference on Uncertainty in Artificial Intelligence*. San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 2003 (UAI'03). – ISBN 0–127–05664–5, 403–410

[MFP00] McCallum, Andrew ; Freitag, Dayne ; Pereira, Fernando C. N.: Maximum Entropy Markov Models for Information Extraction and Segmentation. In: *Proceedings of the Seventeenth International Conference on Machine Learning*. San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 2000 (ICML '00). – ISBN 1–55860–707–2, 591–598

[Mit97] Mitchell, Thomas M.: *Machine Learning*. 1. New York, NY, USA : McGraw-Hill, Inc., 1997. – ISBN 0070428077, 9780070428072

[NJ01] Ng, Andrew Y. ; Jordan, Michael I.: *On Discriminative vs. Generative classifiers: A comparison of logistic regression and naive Bayes*. 2001

[Pea82] Pearl, J: Reverend Bayes on inference engines: A distributed hierarchical approach, 1982, S. 133–136

[Rab89] Rabiner, L.: A tutorial on hidden Markov models and selected applications in speech recognition. In: *Proceedings of the IEEE* 77 (1989), Nr. 2, S. 257–286. http://dx.doi.org/10.1109/5.18626. – DOI 10.1109/5.18626. – ISSN 0018–9219

[RM51] Robbins, Herbert ; Monro, Sutton: A Stochastic Approximation Method. In: *The Annals of Mathematical Statistics* 22 (1951), Nr. 3, 400–407. http://dx.doi.org/10.1214/aoms/1177729586. – DOI 10.1214/aoms/1177729586. – ISSN 0003–4851, 2168–8990. – Mathematical Reviews number (MathSciNet) MR42668, Zentralblatt MATH identifier0054.05901

[RSB12] Roux, Nicolas L. ; Schmidt, Mark ; Bach, Francis: A Stochastic Gradient Method with an Exponential Convergence Rate for Finite Training Sets. Version: 2012. http://arxiv.org/abs/1202.6258. 2012 (1202.6258). – arXiv e-print

[Say86] Sayers, Michael W.: The International Road Roughness Experiment (IRRE) : establishing correlation and a calibration standard for measurements / The World Bank. 1986 (WTP45). – Forschungsbericht. – 1–468 S.

[Say98] Sayers, M. W.: The little book of profiling: basic information about measuring and interpreting road profiles. (1998), September. http://deepblue.lib.umich.edu/handle/2027.42/21605. – http://deepblue.lib.umich.edu/bitstream/2027.42/21605/1/90151.pdf

[Sch99] Schraudolph, Nicol N.: Local Gain Adaptation in Stochastic Gradient Descent / Istituto Dalle Molle Di Studi Sull Intelligenza Artificiale. 1999. – Forschungsbericht

[Sch02] Schraudolph, Nicol N.: Fast Curvature Matrix-Vector Products for Second-Order Gradient Descent. In: *Neural Computation* 14 (2002), S. 2002

[Sei14] Seifen, Sebastian: *A Mathematical Model for Grouped Extreme Values with an Application in Automotive Engineering.* Fraunhofer IRB Verlag, 2014. – ISBN 9783839607015

[Sha70] Shanno, D. F.: Conditioning of quasi-Newton methods for function minimization. In: *Mathematics of Computation* 24 (1970), Nr. 111, 647–656. http://dx.doi.org/10.1090/S0025-5718-1970-0274029-X. – DOI 10.1090/S0025–5718–1970–0274029–X

[Sha11] Shashanka, Madhusudana: *A Fast Algorithm for Discrete Hmm Training Using Observed Transitions.* 2011

[SL+14] Speckert, Michael ; Lemken, Alexander u. a.: *Virtual Measurement Campaign - Report.* 2014

[SM50] Sherman, Jack ; Morrison, Winifred J.: Adjustment of an Inverse Matrix Corresponding to a Change in One Element of a Given Matrix. In: *The Annals of Mathematical Statistics* 21 (1950), März, Nr. 1, 124–127. http://www.jstor.org/stable/2236561. – ISSN 0003–4851

[SM05] Sutton, Charles ; McCallum, Andrew: Piecewise Training for Undirected Models. In: *arXiv:1207.1409 [cs, stat]* (2005). http://arxiv.org/abs/1207.1409. – arXiv: 1207.1409

[SM07] Sutton, C ; Mccallum, A: Piecewise Pseudolikelihood for Efficient Training of Conditional Random Fields, 2007, S. 863–870

[SM10] Sutton, Charles ; McCallum, Andrew: An Introduction to Conditional Random Fields. Version: November 2010. http://arxiv.org/abs/1011.4088. 2010 (1011.4088). – arXiv e-print

[Sta95] Standardization, ISO International Organization f.: Mechanical vibration - road surface profiles - reporting of measured data, ISO 8608:1995(E). In: *Statistics & Probability Letters* (1995)

[STVS09] Sunehag, Peter ; Trumpf, Jochen ; Vishwanathan, S. V. N. ; Schraudolph, Nicol: Variable Metric Stochastic Approximation Theory. In: *arXiv:0908.3529 [physics]* (2009), August. http://arxiv.org/abs/0908.3529. – Proceedings of AISTATS 2009 Clearwater Florida, Volume 5. of JMLR: W&CP 5

[SYG07] Schraudolph, Nicol N. ; Yu, Jin ; Günter, Simon: A stochastic Quasi-Newton Method for online convex optimization. In: *In Proceedings of 11th International Conference on Artificial Intelligence and Statistics*, 2007

[Tib11] Tibshirani, Robert: Regression shrinkage and selection via the lasso: a retrospective. In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 73 (2011), Nr. 3, 273–282. http://onlinelibrary.wiley.com/doi/10.1111/j.1467-9868.2011.00771.x/full

[TMF04] Torralba, Antonio ; Murphy, Kevin P. ; Freeman, William T.: Contextual models for object detection using boosted random fields. In: *Advances in neural information processing systems*, 2004, 1401–1408

[Vap] Vapnik, Vladimir N.: *The Nature of Statistical Learning Theory*. Springer New York

[Wol71] Wolfe, P.: Convergence Conditions for Ascent Methods. II: Some Corrections. In: *SIAM Review* 13 (1971), April, Nr. 2, 185–188. http://dx.doi.org/10.1137/1013035. – DOI 10.1137/1013035. – ISSN 0036–1445

[ZH05] Zou, Hui ; Hastie, Trevor: Regularization and variable selection via the elastic net. In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 67 (2005), April, Nr. 2, 301–320. http://dx.doi.org/10.1111/j.1467-9868.2005.00503.x. – DOI 10.1111/j.1467–9868.2005.00503.x. – ISSN 1467–9868

Scientific Career

Personal Data

Surname	Lemken
First Name	Alexander

Scientific Education

10/2003-10/2010	Studies in computer science at RWTH Aachen - Minor field of study: mathematics - Special subject: computer graphics - Degree: Diplom (Diplom-Informatiker, Dipl.-Inform)
10/2003-03/2011	Studies in mathematics at RWTH Aachen - Minor field of study: physics - Special subject: stochastics, time series - Degree: Diplom (Diplom-Mathematiker, Dipl.-Math)
10/2011-03/2015	Doctoral studies at Technical University Kaiserslautern, Department of Mathematics, Statistics Group
10/2011-03/2015	Doctoral scholarship from Fraunhofer Society Scholar at Fraunhofer Institue for Industrial Mathematics ITWM in Kaiserslautern, department Mathematical Methods in Dynamics and Durability MDF

Wissenschaftlicher Werdegang

Persönliche Daten

Nachname	Lemken
Vorname	Alexander

Wissenschaftlicher Werdegang

10/2003-10/2010	Studium der Informatik an der RWTH Aachen - Nebenfach: Mathematik - Vertiefung: Computergrafik - Abschluss: Diplom (Diplom-Informatiker, Dipl.-Inform)
10/2003-03/2011	Studium der Mathematik an der RWTH Aachen - Nebenfach: Physik - Vertiefung: Stochastik, Zeitreihen - Abschluss: Diplom (Diplom-Mathematiker, Dipl.-Math)
10/2011-03/2015	Promotionsstudium an der TU Kaiserslautern, Fachbereich Mathematik, Arbeitsgruppe Statistik
10/2011-03/2015	Promotionsstipendium der Fraunhofer Gesellschaft Promotionsstudent am Fraunhofer Institut für Techno- und Wirtschaftsmathematik, ITWM in Kaiserslautern Abteilung Mathematische Methoden in Dynamik und Festigkeit, MDF